WebSphere MQ for z/VSE

WebSphere MQ for z/VSE Version 3.0.0 System Management Guide

Version 3 Release 0 Modification 0

GC34-6981-04

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 1091.

Fifth edition (January 2013)

This edition applies to WebSphere MQ for z/VSE Version 3 Release 0 Modification 0 and to any subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories, Information Development,
Mail Point 095, Hursley Park, Winchester, Hampshire, England,
SO21 2JN

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

About this book

WebSphere® for Version 3.0.0—referred to in this book as WebSphere MQ for z/VSE® or simply WebSphere MQ, as the context permits—is part of the WebSphere MQ family of products. These products provide application programming services that enable application programs to communicate with each other using *message queues*. This form of communication is referred to as *commercial messaging*. The applications involved can exist on different nodes on a wide variety of machine and operating system types. They use a common application programming interface, called the Message Queuing Interface or MQI, so that programs developed on one platform can be readily transferred to another.

This book describes the system administration aspects of WebSphere MQ for z/VSE Version 3.0.0 and the services it provides to support commercial messaging in a z/VSE environment. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Who this book is for

Primarily, this book is for system administrators, and system programmers who manage the configuration and administration tasks for WebSphere MQ. It is also useful to application programmers who must have some understanding of WebSphere MQ administration tasks.

What you need to know to understand this book

To use this book, you should have a good understanding of the z/VSE operating system, and utilities associated with it. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

How to use this book

Read Chapter 1, "Introduction," on page 1 first for an understanding of WebSphere MQ for z/VSE.

The body of this book contains:

At the back of the book there are some appendixes giving information (which will be incorporated in the appropriate WebSphere MQ books at the next opportunity) on these topics:

Summary of changes

This section describes changes to this edition of the *WebSphere MQ for z/VSE System Management Guide*.

Changes since the last edition of the *MQSeries for VSE System Management Guide* are marked by vertical lines to the left of the changes.

Changes in this edition (GC34-6981-04)

The changes in this edition of the System Management Guide are updates and additions to describe the new features and improvements associated with WebSphere MQ for z/VSE V3.0.0.

In addition to minor changes throughout the manual, the major additions and modifications to this edition include:

- The WebSphere MQ transport for SOAP provides a JMS transport for SOAP

  A SOAP client using Apache Axis 1.4 platform can send a web service request in a SOAP envelope to a z/VSE SOAP server using WebSphere MQ.

  For more information, see Appendix I, "WMQZVSE SOAP transport to z/VSE SOAP server," on page 1071.
- Publish/Subscribe

  Publish/subscribe messaging allows you to decouple the provider of information from the consumers of that information. The sending application and receiving application do not need to know anything about each other for the information to be sent and received.

  Before a point-to-point WebSphere MQ application can send a message to another application, it needs to know something about that application. For example, it needs to know the name of the queue to which to send the information, and might also specify a queue manager name.

  For more information see Appendix J, "Publish/Subscribe," on page 1075.
- Channel Authentication

  To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

  For more information see "Inquire Channel Authentication Records" on page 308, "Set Channel Authentication Record" on page 345, "WebSphere MQ channel authentication" on page 532, and Appendix K, "Channel authentication records," on page 1087.

Changes in GC34-6981-03

The changes in this edition of the System Management Guide are updates and additions to describe the new features and improvements associated with WebSphere MQ for z/VSE V3.0.0.

In addition to minor changes throughout the manual, the major additions and modifications to this edition include:

- **Message properties**

Use message properties to allow an application to select messages to process or to retrieve information about a message without accessing MQMD or MQRFH2 headers. Message properties also facilitate communication between Websphere MQ and JMS applications.

A message property is data associated with a message, consisting of a textual name and a value of a particular type. You can use message properties to include business data or state information without having to store it in the application data. Applications do not have to access data in the MQ Message Descriptor (MQMD) or MQRFH2 headers because fields in these data structures can be accessed as message properties using Message Queue Interface (MQI) function calls.

The use of message properties in WebSphere MQ mimics the use of properties in JMS. This means that you can set properties in a JMS application and retrieve them in a procedural WebSphere MQ application, or the other way around.

Changes in GC34-6981-02

The changes in this edition of the System Management Guide are updates and additions to describe the new features and improvements associated with WebSphere MQ for z/VSE V3.0.0.

In addition to minor changes throughout the manual, the major additions and modifications to this edition include:

- **Command, Configuration, and SSL events**

  Command events are notifications that an MQSC or PCF command has run successfully.

  Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

  SSL events are a type of channel event. 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.

  For more information, see “Instrumentation events” on page 221.
- **Listener and Service objects.**

  For more information, see:
  – “WebSphere MQ channel listener” on page 539.
  – “WebSphere MQ Service” on page 589.
- **New WHERE keyword**

  The WHERE keyword is provided for the MQSC DISPLAY commands, and integer filter structure (MQCFIF), string filter structure (MQCFSF) for the PCF inquire commands. This allows you to filter the information displayed by one (and only one) of the attributes objects.

  For more information, see “WHERE” on page 511.
- **Support for message monitoring.**

  For more information, see “Message monitoring” on page 149.

Changes in GC34-6981-01

The changes in this edition of the System Management Guide are updates and additions to describe the new features and improvements associated with WebSphere MQ for z/VSE V3.0.0.

In addition to minor changes throughout the manual, the major additions and modifications to this edition include:

- **Accounting and statistics message generation**

  Accounting and statistics messages are generated intermittently by queue managers to record information about the MQI operations performed by WebSphere MQ applications, or to record information about the activities occurring in a WebSphere MQ system. Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications. Statistics messages are used to record information about the activities occurring in a WebSphere MQ system.

  Refer to "Accounting and statistics messages" on page 434 for more information.

- **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. For more information refer to "Real-time monitoring" on page 482.

  For information about real-time monitoring PCF commands refer to "Inquire Channel Status" on page 315 and "Inquire Queue Status" on page 337.

  For information about real-time monitoring MQSC commands refer to "DISPLAY CHSTATUS" on page 521 and "DISPLAY QSTATUS" on page 572.

- **SSL key reset**

  SSL-enabled channels negotiated a secret key used to encrypt and decrypt data sent over a channel. For long running channels, this may present a security exposure as the secret key may be discovered and used to view or modify encrypted transmissions. For this reason, WebSphere MQ for z/VSE now supports an SSL key reset feature where by the key can be renegotiated after a configurable number of bytes have flowed over the channel.

  Refer to "SSL reset count" on page 645 for more information.

- **Connection commands**

  WebSphere MQ for z/VSE now supports connection commands. Connection commands allow you to view information about active connections to the queue manager, and to stop connections. Support extends to both PCF and MQSC commands.

  For PCF, refer to "Inquire Connection" on page 322 and "Stop Connection" on page 353.

  For MQSC, refer to "WebSphere MQ connection commands" on page 544.

Changes in GC34-6981-00

The changes in this new edition of the System Management Guide are updates and additions to describe the new features and improvements associated with WebSphere MQ for z/VSE, including the rebranding of MQSeries® to WebSphere MQ, and the progression of VSE to z/VSE.

In addition to minor changes throughout the manual, the major additions and modifications to this edition are:

- **API exits:** API exits let you write code that changes the behavior of WebSphere MQ API calls, such as MQPUT and MQGET, and then insert that code immediately before or immediately after those calls. The insertion is automatic;

the queue manager drives the exit code at the registered points. On z/VSE, WebSphere MQ supports a chain of up to eight API exits. API exits are described in Chapter 13, "API exits," on page 677.

- **WebSphere MQ Explorer:** WebSphere MQ for Windows and WebSphere MQ for Linux (x86 platform) include an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. 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. This interface is enhanced to support remote administration of both MQSeries for VSE V2.1.2 and WebSphere MQ for z/VSE V3.0. WebSphere MQ Explorer is described in "Administration using the WebSphere MQ Explorer" on page 154.
- **Server and requester channels:** Server and requestor channels are additional channel types that allow queue managers to request messages from remote systems, rather than wait for those systems to activate the flow. This means messages can accumulate on a remote system until they are needed by the queue manager. For more information about server and requester channels, refer to "Message channels" on page 7.
- **Chained message exits:** Channel exit programs are called at defined places in the processing carried out by Message Channel Agent (MCA) programs. These are the communications that facilitate remote queuing to other queue managers and client connectivity. On z/VSE, WebSphere MQ now supports a chain of up to eight send, receive, and message exits. Refer to "Channel exits" on page 55 for more information.

Chapter 1. Introduction

This chapter introduces WebSphere MQ for z/VSE from an administrator's perspective, and describes the basic concepts of WebSphere MQ and messaging.

WebSphere MQ and message queuing

WebSphere MQ lets z/VSE applications use message queuing to participate in message-driven processing. Applications can communicate across different platforms by using the appropriate message queuing software products. For example, z/VSE and z/OS® applications can communicate through WebSphere MQ for z/VSE and WebSphere MQ for z/OS respectively. The applications are shielded from the mechanics of the underlying communications.

WebSphere MQ products implement a common application programming interface (message queue interface or MQI) whatever platform the applications are run on. This makes it easier to port applications from one platform to another.

The MQI is described in detail in the *WebSphere MQ Application Programming Reference* manual and Appendix B, "Application Programming Reference," on page 717.

Time-independent applications

With message queuing, the exchange of messages between the sending and receiving programs is time independent. This means that the sending and receiving applications are decoupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of the message. In fact, the target application does not even have to be running when the message is sent. It can retrieve the message after it is started.

Message-driven processing

Applications can be automatically started by messages arriving on a queue using a mechanism known as *triggering*. If necessary, the applications can be stopped when the message or messages have been processed.

Synchronous applications

WebSphere MQ also provides for synchronous applications. An application can wait for a message to arrive on a queue. For example, in a client/server environment, a client application can place a message on a request queue (one designated for client requests), and then wait for a response message on a reply queue (one designated for server replies). The server application can wait for request messages and send reply messages as requests are processed.

Messages and queues

Messages and queues are the basic components of a message queuing system.

What messages are

A *message* is a string of bytes that has meaning to the applications that use it. Messages are used for transferring information from one application to another (or to different parts of the same application). The applications can be running on the same platform, or on different platforms.

WebSphere MQ messages have two parts; the *application data* and a *message descriptor*. The content and structure of the application data is defined by the application programs that use them. The message descriptor identifies the message and contains other control information, such as the type of message and the priority assigned to the message by the sending application.

The format of the message descriptor is defined by WebSphere MQ for z/VSE. For a complete description of the message descriptor, see the *WebSphere MQ Application Programming Reference* manual.

Message lengths

In WebSphere MQ for z/VSE, the maximum message length is 4 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length may be limited by:

- The maximum message length defined for the receiving queue.
- The maximum message length defined for the queue manager.
- The maximum message length defined by either the sending or receiving application.
- The amount of storage available for the message.

  This parameter is extremely important for WebSphere MQ for z/VSE. The storage will be used from the CICS® partition in which the queue manager is active.

It may take several messages to send all the information that an application requires.

What queues are

A *queue* is a data structure that stores zero or more messages. The messages may be put on the queue by applications or by a queue manager as part of its normal operation.

Each queue belongs to a *queue manager*, which is responsible for maintaining it. The queue manager puts the messages it receives on the appropriate queues.

Applications send and receive messages using MQI calls. For example, one application can put a message on a queue, and another application can retrieve the message from the same queue. The sending application opens the queue for put operations by making an MQOPEN call. Then it issues an MQPUT call to put the message onto that queue. When the receiving application opens the same queue for gets, it can retrieve the message from the queue by issuing an MQGET call.

For more information about MQI calls, see the *WebSphere MQ Application Programming Reference* manual.

WebSphere MQ for z/VSE supports predefined and dynamic queues:

- Predefined queues are those created by an administrator using the appropriate command set, for example, those defined using the WebSphere MQ Master

Terminal (MQMT) utility. Predefined queues are permanent; they exist independently of the applications that use them and survive WebSphere MQ for z/VSE restarts.

- Dynamic queues are created by applications. An application can create a queue using the MQOPEN MQI call. Queues created this way can be temporary or permanent. A temporary dynamic queue is removed from the queue manager when the application that created it closes the queue or terminates. A permanent dynamic queue can be closed but not removed, and like predefined queues, permanent dynamic queues can exist independently of the applications that use them and survive WebSphere MQ for z/VSE restarts. Alternatively, an application can choose to remove a permanent dynamic queue from the queue manager using the MQCLOSE MQI call.

Retrieving messages from queues

In WebSphere MQ for z/VSE, suitably authorized applications can retrieve messages from a queue according to these retrieval algorithms:

- First-in-first-out (FIFO).
- A program request for a specific message, identified by a message identifier or correlation identifier.
- A program request for a specific message, identified by a combination of group name and message sequence number. Getting messages in this way is described in "Message grouping and segmentation" on page 959.

The MQGET request from the application determines the method used.

Objects

Many of the tasks described in this document involve manipulating WebSphere MQ *objects*. In WebSphere MQ for z/VSE, there are six different types of object:

- Queue managers; see "WebSphere MQ queue managers" on page 4.
- Queues; see "WebSphere MQ queues" on page 5.
- Channels; see "Channels" on page 7.
- Namelists; see "Namelists" on page 9.
- Listeners; see "Listeners" on page 10.
- Services; see "Services" on page 10.

Object names

Each instance of a queue manager is known by its name. This name must be unique within the network of interconnected queue managers, so that one queue manager can unambiguously identify the target queue manager to which any given message should be sent.

For the other types of object, each object has a name associated with it and can be referenced in WebSphere MQ for z/VSE by that name. These names must be unique within one queue manager and object type. For example, you can have a queue and a channel with the same name, but you cannot have two queues with the same name.

In WebSphere MQ, names can have a maximum of 48 characters, with the exception of *channel names*, which have a maximum of 20 characters. Object names can contain alphanumeric characters or any of the following special symbols:

% Percent
_ Underscore
. Dot

/ Slash

Object names should not contain leading or embedded spaces.

Managing objects

WebSphere MQ provides commands for creating, altering, displaying, and deleting objects through the panel driven WebSphere MQ Master Terminal (MQMT) system administration transaction; see "WebSphere MQ master terminal (MQMT) – main menu" on page 79 for further details.

You can perform some limited administration, for example, the starting and stopping of queues and channels, by using the MQCL transaction. See Chapter 5, "Utilities and interfaces," on page 169 for further details.

Local and remote administration

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 TCP/IP, 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.

Remote administration of WebSphere MQ for z/VSE is possible using Programmable Command Format (PCF) messages. These are special binary messages that can be used to instruct the queue namager to create, modify, or delete objects owned by the queue manager. An administrative application running on a remote system can send PCF message to a special queue called the system command queue for this purpose. Programmable command formats are described in detail in Chapter 8, "Programmable system management," on page 221.

Remote administration is also possible using the WebSphere MQ Explorer interface which is available for Windows and Linux (x86). For more information about remote administration using the Explorer, refer to "Administration using the WebSphere MQ Explorer" on page 154.

Object attributes

The properties of an object are defined by its attributes. Some you can specify, others you can only view. For example, the maximum message length that a queue can accommodate is defined by its *MaxMsgLength* attribute (see Figure 33 on page 101). You can specify this attribute when you create a queue.

In WebSphere MQ for z/VSE, there are five ways of accessing an attribute:

- Using the MQMT transaction, described in "WebSphere MQ master terminal (MQMT) – main menu" on page 79.
- Using the MQINQ function call, described in "MQINQ - Inquire about object attributes" on page 893.
- Using Programmable Command Format (PCF) messages, described in Chapter 8, "Programmable system management," on page 221.
- Using WebSphere MQ Commands (MQSC), described in Chapter 9, "WebSphere MQ commands," on page 507.
- Using the WebSphere MQ Explorer, described in "Administration using the WebSphere MQ Explorer" on page 154.

WebSphere MQ queue managers

Queues are defined to WebSphere MQ using the MQMT master terminal transaction, PCF requests, WebSphere MQ commands, or the WebSphere MQ

Explorer. Each type of queue has the same attributes but its own value for the attributes. For example, a local queue definition specifies:

- Object attributes are changed according to the commands received.
- Special events such as trigger events are generated when the appropriate conditions are met.
- Messages are put on the correct queue, as requested by the application making the MQPUT call. The application is informed if this cannot be done, and an appropriate reason code is given.
- Messages destined for remote queue managers are sent and message from remote queue managers are received.

Each queue belongs to a single queue manager and is said to be a *local queue* to that queue manager. The queue manager to which an application is connected is said to be the local queue manager for that application. For the application, the queues that belong to its local queue manager are local queues.

A *remote queue* is simply a queue that belongs to another queue manager. A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager exists on a remote machine across the network, or in a different CICS region on the same z/VSE host.

MQI calls

A queue manager object may be used in some MQI calls. For example, you can inquire about the attributes of the queue manager object using the MQI call MQINQ.

Note: You cannot put messages on a queue manager object; messages are always put on queue objects, not on queue manager objects.

WebSphere MQ queues

Queues are defined to WebSphere MQ using the appropriate MQMT transaction, via PCF requests, or via the WebSphere MQ Explorer. The transaction specifies the type of queue and its attributes. For example, a local queue object has attributes that specify what happens when applications reference that queue in MQI calls. Examples of attributes are:

- Whether applications can retrieve messages from the queue (GET enabled).
- Whether applications can put messages on the queue (PUT enabled).
- Whether access to the queue is exclusive to one application or shared between applications.
- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth).
- The maximum length of messages that can be put on the queue.

Using queue objects

In WebSphere MQ for z/VSE, there are four types of queue object. Each type of object can be manipulated by the product commands and is associated with real queues in different ways:

1. A *local queue* object identifies a local queue belonging to the queue manager to which the application is connected. All queues are local queues in the sense that each queue belongs to a queue manager and, for that queue manager, the queue is a local queue.
2. A *remote queue object* identifies a queue belonging to another queue manager. This queue must be defined as a local queue to that queue manager. The

information you specify when you define a remote queue object allows the local queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.

You must also define a transmission queue and channels between the queue managers, before applications can send messages to a queue on another queue manager. A transmission queue is a special type of local queue.

3. An *alias queue object* allows applications to access a queue by referring to it indirectly in MQI calls. When an alias queue name is used in an MQI call, the name is resolved to the name of either a local or a remote queue at run time. This allows you to change the queues that applications use without changing the application in any way—you merely change the alias queue definition to reflect the name of the new queue to which the alias resolves.

   An alias queue is not a queue, but an object that you can use to access another queue.

4. A *model queue object* is a template of a queue definition, that you use when creating a dynamic queue. When an application opens a model queue, the queue manager dynamically creates a local queue with the attributes of the model queue. Consequently, a model queue never contains messages, it is only used as a template to create dynamic queues.

Specific local queues used by WebSphere MQ

WebSphere MQ uses some local queues for specific purposes related to its operation. You ***must*** define them before WebSphere MQ can use them.

Application queues: A queue that is used by an application (through the MQI) is referred to as an *application queue.* This can be a local queue on the queue manager to which an application is linked, or it can be a remote queue that is owned by another queue manager.

Applications can put messages on local or remote queues. However, they can only get messages from a local queue.

Transmission queues: A *transmission queue* temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. For information about the use of transmission queues in distributed queuing, see the *WebSphere MQ Intercommunication* book.

Dead-letter queue: A *dead-letter queue* stores messages that cannot be routed to their correct destinations. This occurs when, for example, the destination queue is full. The supplied dead-letter queue is called SYSTEM.DEAD.LETTER.QUEUE. These queues are also referred to as undelivered-message queues on other platforms.

For distributed queuing, you should define a dead-letter queue for each queue manager.

Event queues: In WebSphere MQ, an instrumentation event is a logical combination of conditions 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. Event queue names are configurable as part of the queue manager's Global System Definition.

Channels

Channels are objects that provide a communication path from one queue manager to another. Channels are used in distributed message queuing to move messages from one queue manager to another. They shield applications from the underlying communications protocols. The queue managers may exist on the same, or different, platforms. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another, complementary one, at the queue manager that is to receive them. Channels can also define a communication path between a remote client and the local z/VSE queue manager. These are called server-connection channels. A complimentary client-connection channel must also be defined in the client environment.

Message channels

Message channels Message channels are the channels that carry messages from one queue manager to another.

Do not confuse message channels with MQI channels. There are two types of MQI channel, server-connection and client-connection. These are discussed in WebSphere MQ Clients.

The definition of each end of a message channel can be one of these types:
- Sender
- Receiver
- Server
- Requester
- Cluster sender
- Cluster receiver

Note that WebSphere MQ for z/VSE does not support cluster-type channels.

A message channel is defined using one of these types defined at one end, and a compatible type at the other end.

Possible combinations are:
- Sender-receiver
- Requester-server
- Requester-sender (callback)
- Server-receiver
- Cluster sender-cluster receiver

To define a channel refer to “Channel definitions” on page 112.

Sender-receiver channels: A sender in one system starts the channel so that it can send messages to the other system. The sender requests the receiver at the other end of the channel to start. The sender sends messages from its transmission queue to the receiver. The receiver puts the messages on the destination queue. Figure 1 on page 8 illustrates this.

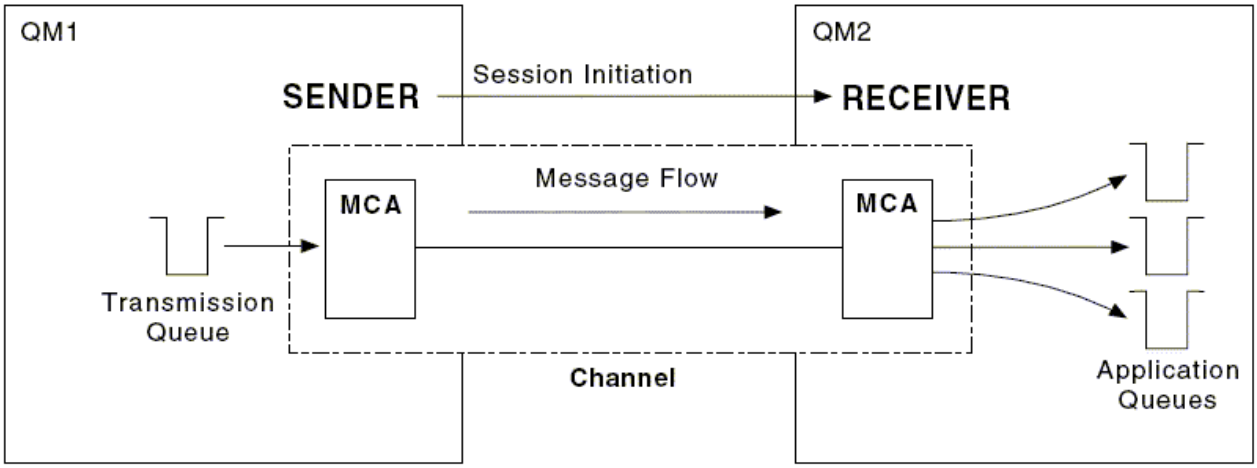


Figure 1. Sender-receiver channels

Requester-server channels: A requester in one system starts the channel so that it can receive messages from the other system. The requester requests the server at the other end of the channel to start. The server sends messages to the requester from the transmission queue defined in its channel definition.

A server channel can also initiate the communication and send messages to a requester, but this applies only to fully qualified servers, that is server channels that have the connection name of the partner specified in the channel definition. A fully qualified server can either be started by a requester, or can initiate a communication with a requester.

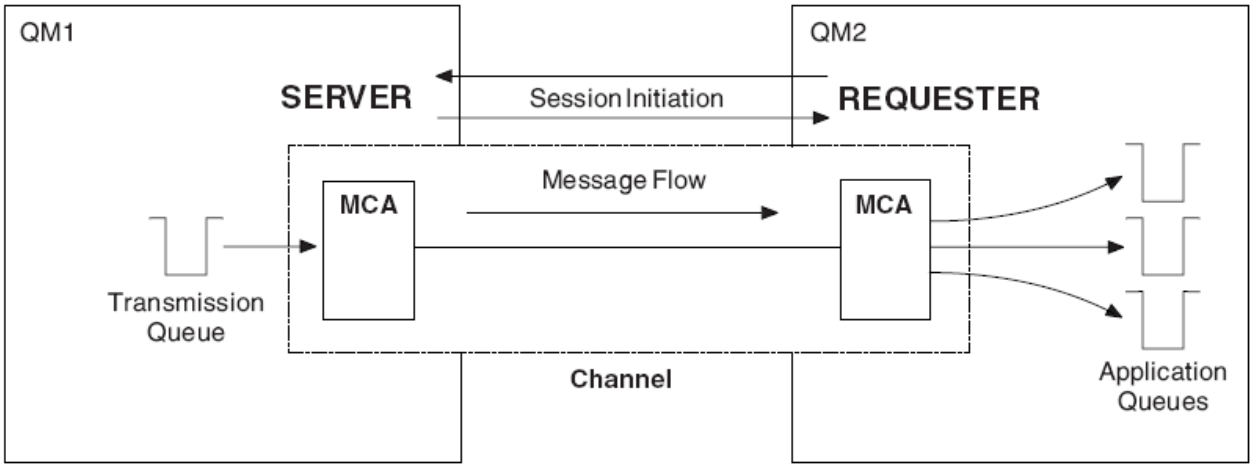


Figure 2. Requester-server channels

Requester-sender channels: The requester starts the channel and the sender terminates the call. The sender then restarts the communication according to information in its channel definition (this is known as callback). It sends messages from the transmission queue to the requester.

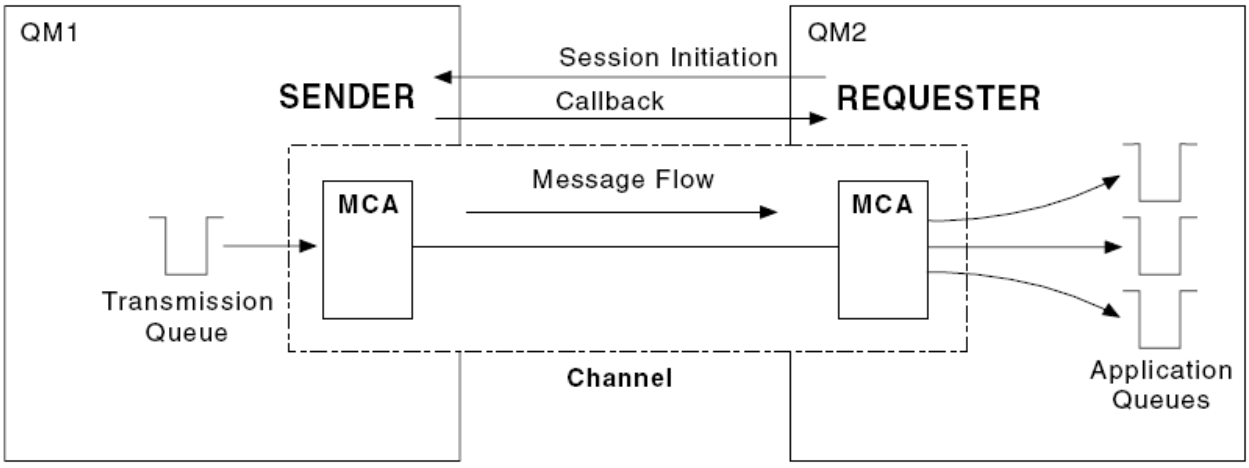


Figure 3. Requester-sender channels

Server-receiver channels: This is similar to sender-receiver but applies only to fully qualified servers, that is server channels that have the connection name of the partner specified in the channel definition. Channel startup must be initiated at the server end of the link.

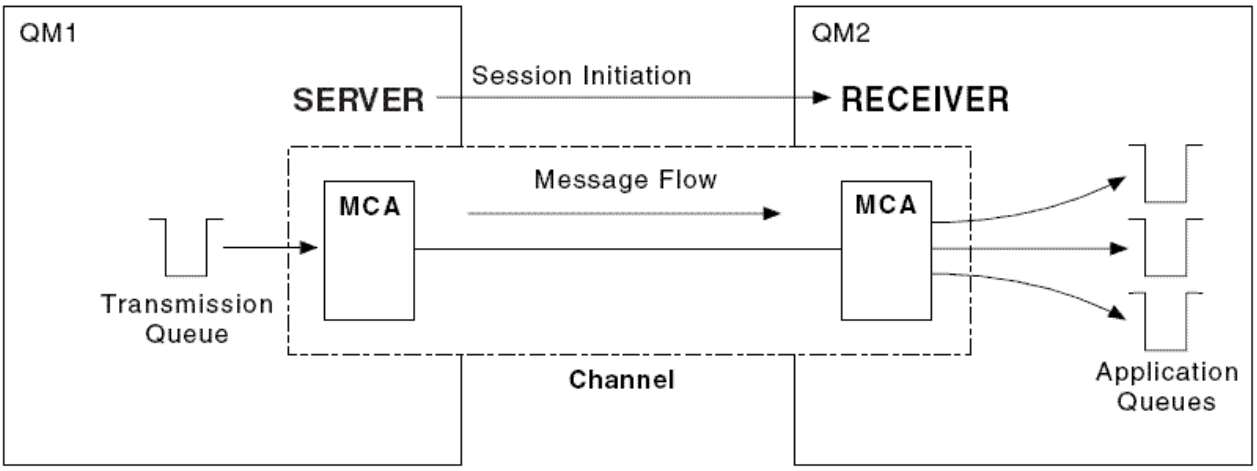


Figure 4. Server-receiver channels

Namelists

A *namelist* is an WebSphere MQ object that contains a list of names, such as queue names. You can define and modify namelists using the WebSphere MQ for z/VSE Master Terminal transactions, that is MQMT, and using PCF or MQSC commands.

Programs can use the MQI to find out which names are included in a namelist. The organization of the namelists is the responsibility of the application designer and system administrator.

An advantage of using a namelist is that it is maintained independently of applications.

Listeners

A listener is a WebSphere MQ object that accepts network requests from other queue managers, or client applications, and starts associated channels. Listener processes can be configured using the master terminal transaction (MQMT), Programmable Command Format (PCF), or MQSeries Command (MQSC) requests.

You can define more than one listener object and select whether the listener is automatically started when the queue manager is started.

Services

A service is a WebSphere MQ object that identifies a user program that is to be started when the queue manager is started. Services fall into two categories:

Servers
A server service object is the definition of a program that is executed when a specified queue manager is started. Only one instance of a server process can be executed concurrently.

Commands
A command service object is the definition of a program that is executed when a specified queue manager is started or stopped. Multiple instances of a command process can be executed concurrently.

Service objects can be created, modified, and deleted using the master terminal transaction (MQMT), Programmable Command Format (PCF), or MQSeries Command (MQSC) requests.

Clients and servers

WebSphere MQ for z/VSE supports client-server configurations for WebSphere MQ applications, and can act as a server to which WebSphere MQ clients can connect. The WebSphere MQ client environment, however, is not part of WebSphere MQ for z/VSE product, but can be installed and used independently. See Chapter 10, "WebSphere MQ clients," on page 623 for more information.

An *WebSphere MQ client* is a part of the WebSphere MQ product that is installed on a machine to accept MQI calls from applications and pass them to an *MQI server* machine. There they are processed by a queue manager. Typically, the client and server reside on different machines but they can also exist on the same machine.

An *MQI server* is a queue manager that provides queuing services to one or more clients. For z/VSE, there is one WebSphere MQ process for each client connection.

All the WebSphere MQ objects, for example queues, exist only on the queue manager machine, that is, on the MQI server machine. A server can support normal local WebSphere MQ applications as well.

The difference between an MQI server and an ordinary queue manager is that a server has a dedicated communications link with each client. For more information about creating channels for clients and servers, see the *WebSphere MQ Intercommunication* book.

WebSphere MQ applications in a client-server environment

When connected to a server, client WebSphere MQ applications can issue MQI calls in the same way as local applications. The client application issues an MQCONN

call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager. You must link your applications to the appropriate client libraries. See the *WebSphere MQ Application Programming Guide* for further information.

WebSphere MQ and CICS

WebSphere MQ for z/VSE runs as a CICS task. Consequently, various features of the product are controlled by CICS itself.

These features include security and recovery. If you install WebSphere MQ for z/VSE with the security feature, security will be handled by your External Security Manager (ESM) or with z/VSE 4.3, the Basic Security Manager (BSM) now has the MQ classes required to support WebSphere MQ for z/VSE security.

Chapter 2. Installation

This chapter describes the procedure for installing WebSphere MQ for z/VSE. It consists of these sections:

Contents of the library tape

The distribution tape is in standard IBM® MSHP format and may be stacked or non-stacked format depending on how the product is ordered but should be handled in the same way by the z/VSE install procedures. The tape will contain a sublibrary for "PRD2.WMQZVSE".

This sublibrary contains:

- Copy books, used by your CICS applications whenever you intend to call the WebSphere MQ Message Queuing Interface (MQI).
- Object decks, called at linkedit time when you are building your own WebSphere MQ applications (autolink).
- Phases, to provide WebSphere MQ operation in CICS and Batch.
- Samples having member type A or Z. Some of these need to be modified for the VSE/POWER JECL statements, as follows:

```
* ** JOB  to   * $$ JOB
* ** LST  to   * $$ LST
* ** SLI  to   * $$ SLI
* ** EOJ  to   * $$ EOJ
```

The samples are:

DCHFMT4.Z
Sample data conversion exit program for message data conversion.

MQBICALL.Z
Sample batch interface program that shows how to write an MQI batch program.

MQBISTOP.Z
Sample program to stop the batch interface from a batch partition.

MQCICDCT.A
Entry definitions for CICS DCT.

MQCICFCT.A
Entry definitions for CICS FCT.

MQDOCU.Z
New documentation that has been instroduced through the service stream, but not yet available in this manual.

MQHTML.Z
HTML source file mapping to the WMQ Master Terminal transactions for use by a remote browser via CICS Web Support.

MQJCONFG.Z
Creation of WebSphere MQ configuration file.

MQJCSD.Z
Define CICS resources into the CICS CSD.

MQJCSD24.Z
Define CICS resources into the CICS CSD for CICS TS customers.

MQJCWS.Z
HTML catalog job for CICS Web Support.

MQJINSG.Z
Sample WebSphere MQ command JCL to create default queues and channels.

MQJLABEL.Z
Label definitions for the CICS start-up job.

MQJMIGR1.Z
Migration of old configuration file (step 1).

MQJMIGR2.Z
Migration of old configuration file (step 2).

MQJQUEUE.Z
VSAM cluster definitions for WebSphere MQ queues.

MQJREORG.Z
Batch job to reclaim space of deleted records.

MQJSETUP.Z
Creation of the setup file.

MQJUTILY.Z
Various batch functions.

MQPECHO.Z
Sample COBOL MQI program designed to run as a trigger program.

MQPSAXE.Z
Sample COBOL MQ API Exit program.

MQSERIES.Z
WebSphere MQ for z/VSE tape header file. For internal use only.

MQUSERID.Z
Sample assembler to allow a change of user identifier for MCA communications with remote AS/400® systems.

SYSIN.Z
Master configuration input file used during installation to create the WMQ for z/VSE configuration file.

TTMTST3.Z
Sample Assembler CICS map program for COBOL sample TTPTST3.

TTPTST1.Z
Sample COBOL MQI program.

TTPTST2.Z
Sample COBOL MQI program.

TTPTST3.Z
Sample COBOL test to drive sample TTPTST2

Prerequisites

Program number

- 5655-U97 WebSphere MQ for z/VSE Version 3 Release 0.0.

Hardware requirements

- WebSphere MQ Servers:

  IBM System z10™ Enterprise Class
  IBM System z10 Business Class
  IBM System z9® Enterprise Class
  IBM System z9 Business Class
  IBM zSeries® 890
  IBM zSeries 990
  IBM zSeries 800
  IBM zSeries 900
  S/390® Parallel Enterprise Server—Generation 5 and 6
  S/390 Multiprise 3000
  Equivalent server

Software requirements

Minimum supported levels are shown. Later levels, if any, will be supported unless otherwise stated. Note that the latest maintenance for these requirements is strongly recommended.

- z/VSE 3.1 or later.
- CICS/VSE 2.3 or CICS TS 1.1, or later.
- VTAM® for z/VSE 4.2 or TCP/IP for z/VSE 1.5F (or equivalent), or later.
- Language Environment® for z/VSE 1.4.4 Runtime library, or later.
- WebSphere MQ Clients:

  WMQ for VSE supports clients that connect using TCP/IP.

Note: As prerequisite software levels become out-of-service, it is strongly recommended that you upgrade to supported levels of all prerequisite software.

Features

The features described in this book are provided with the WebSphere MQ for z/VSE product, with the exception of the WebSphere MQ client which is available as an IBM SupportPAC. Some features, however, are enhancements to the product, and are available only after the relevant APAR/PTFs, or associated service levels, have been applied.

The following list indicates the APAR prerequisites for certain enhancement features:

- WebSphere MQ Explorer support requires one of:
  - WebSphere MQ Explorer V6.0.2.6, or later.
  - WebSphere MQ Explorer V7.0.0.1, or later.
  - WebSphere MQ Explorer Supportpac MS0T.

- SSL key reset requires PK84111.
- Accounting and statistics messages requires PK94386.
- Real-time monitoring requires PM01079.
- PCF and MQSC connection commands requires PM03429.
- Command, configuration and SSL events requires PM09189.
- Listener and service object support requires PM16320.
- Command filtering requires PM23573.
- Message monitoring support requires PM29937.
- Message properties support requires PM48873. In addition, if MQ Explorer is being used, then IC79103 is required.
- Publish/Subscribe support requires PM73453.
- Channel Authentication Record support requires PM78239.

Connectivity

Network protocols supported are SNA LU 6.2 and TCP/IP.
- For SNA connectivity – VTAM for z/VSE, or later.
- For TCP/IP connectivity – TCP/IP for z/VSE V1.5F (or equivalent), or later.

Client connectivity is only available using TCP/IP.

Compilers supported for WebSphere MQ for z/VSE applications

- Programs can be written using C, COBOL or PL/I.
- C programs can use the C for VSE V1.1 compiler, or later.
- COBOL programs can use the COBOL for VSE compiler V1.1, or later.
- PL/I programs can use the PL/I for VSE compiler V1.1, or later.

Delivery

WebSphere MQ for z/VSE is available on:
- 3590 cartridge
- 3592 cartridge

Installing WebSphere MQ for z/VSE - all users

To install the product, carry out the following procedure:

1. Decide the name of the :
   - Target sublibrary

     The target sublibrary can be the default supplied, “PRD2.WMQZVSE”, or a name that you specify.

     If you use the supplied default sublibrary, go to step 2 on page 17.

     If you specify your own library, you must customize the JCL listed in step 1b.
   - VSAM catalog into which the product is to be installed

   a. Create a VSAM user catalog.

      You are recommended to use the Interactive Interface Dialogs (II) to create this catalog. In the following examples, the VSAM catalog named MQMCAT is used, and it is assumed that its label is already defined in the disk label area.

   b. Allocate a z/VSE library.

This step is not required if you restore the product into the PRD2 library. However, if you want to install WebSphere MQ in another library, you must create one. You are recommended to use the Interactive Interface dialogs for creating this library, or run the following sample adapted for your environment.

If you adapt this sample you must modify the sample provided in section 2.b. to use the same sublibrary name.

- `DEFINE S=lib.sublib` to the your selected name

```
* $$ JOB JNM=MQMSUBL,CLASS=O,DISP=D
// JOB MQMSUBL Define the WebSphere MQ installation library
// DLBL mylib,'l.f.i',yyyy/ddd
// EXTENT ,volume,,,n,m
// EXEC LIBR
DEFINE L=mylib
DEFINE S=mylib.sublib
/*
/&
```

where:

mylib is the new library name
sublib is the new sublibrary name
l.f.i is your local file id
yyyyyy/ddd
is the file retention year and day
volume is the local disk volume name
n/m is the start track and size required

See the IBM z/VSE System Control Statements documentation for further information about DLBL, EXTENT and LIBR.

2. Restore the WebSphere MQ sublibrary from the library tape. You can do this by either:
   a. Using the Interactive Interface Dialogs, as follows:
      1) From an administrator ICCF signon, select the “Installation” option.
      2) Select “Install Programs - V2 format”.
      3) Select “Prepare for installation”.

         This presents you with a series of panels and options to identify the tape address and process a job, by scanning the mounted tape and identifying which stacked products are available for installation.

         Monitor the z/VSE console to see when this job has completed. When it has completed, proceed to the step 2a4.
      4) Select “Install Program(s) from Tape”.

         You are presented with a list of products available from the install tape and suggested install sublibraries. You can select either the default install library, “PRD2.WMQZVSE”, or the name of the customized library you created in Step 1 on page 16.
      5) Select option 1 to proceed with the installation and press function key five (PF5) to create a job to be submitted.

      or

   b. Customizing and processing the following JCL, using the library name from step 1 on page 16.

```
* $$ JOB JNM=MQMTAPE,CLASS=0,DISP=D
// JOB MQMTAPE Restore WebSphere MQ from tape
// ASSGN SYS006,cuu
// MTC REW,SYS006
// EXEC MSHP,SIZE=1M
```

```
INSTALL PRODUCT FROMTAPE ID='WMQZVSE....3.0.0' -
PROD INTO=lib.sublib
/*
/&
* $$ EOJ
```

Where:

cuu Is the tape drive address

lib.sublib
Is the sublibrary into which the product is to be installed, for example, PRD2.WMQZVSE

Installation checkpoint (WebSphere MQ installation)

You should now have correctly installed the WebSphere MQ sublibrary. This can be verified using a z/VSE Librarian job to inspect the contents of the library.

The WebSphere MQ phases, objects, and sample jobs are visible.

Note: If the WebSphere MQ product has not installed correctly, check through the preceding instructions to ensure that they all completed correctly.

If you are a new user, see "Procedures for new users." If you are migrating to WebSphere MQ for z/VSE V3.0.0 from an earlier release, see "Migration procedures for existing users" on page 39.

Procedures for new users

The following steps describe how to
- Allocate and initialize the required WebSphere MQ files.
- Customize your CICS system to utilize the WebSphere MQ facilities.

The samples for the following jobs can be found in the installation library you selected, or "PRD2.WMQZVSE".

Allocate and initialize the required WebSphere MQ files

You must now run the jobs to:
- Create the setup file.
- Create the WebSphere MQ configuration file.
- Create cluster definitions for WebSphere MQ queues.

The sample JCL jobs **must** be modified and customized to refer to your own volume identifiers and catalog names.

This should be done by your z/VSE systems programmer.

MQJSETUP.Z
Allocate a VSAM ESDS, MQFSSET, which is needed to populate the WebSphere MQ configuration file with text and help messages at initialization time.

Note: Review the section "Installing security" on page 19 before running this sample JCL.

MQJCONFG.Z
Allocates the WebSphere MQ (CICS) subsystem configuration file. For this VSAM KSDS file, each record is a fixed length of approximately 2 KB.

To estimate the space you require, allocate one record, consisting of one cylinder for normal operation, for each WebSphere MQ channel and queue.

MQJQUEUE.Z

Allocates and initializes the WebSphere MQ message queue files. For these VSAM KSDS files, each record is of varying length, depending upon the size of the user data area. A message queue file is required for each queue defined to the WebSphere MQ (CICS) subsystem.

To estimate the space required for each message queue, use the following guidelines:

- Each message queue file contains one header record for each local queue.
- One record is written for each user message.
- Each record is of variable length and consists of a header of 736 bytes plus the actual variable-length user data area.
- This job allocates the following system queue files:
  WMQZVSE.MQFERR - Dead letter queue file.
  WMQZVSE.MQFLOG - Error log queue file.
  WMQZVSE.MQFMON - Monitor queue file.
  WMQZVSE.MQFREOR - Automatic VSAM reorganization file.
  WMQZVSE.MQFADMN - PCF and MQSC queue file.
  WMQZVSE.MQFDEFS - WebSphere MQ Explorer model queue file.
  WMQZVSE.MQFACCTS - WebSphere MQ accounting message file.
  WMQZVSE.MQFSTATS - WebSphere MQ statistics message file.

  and optionally the following files:
  WMQZVSE.MQFACMD - Admin command file.
  WMQZVSE.MQFARPY - Admin reply file.
  WMQZVSE.MQFIEQE - Queue manager events file.
  WMQZVSE.MQFIECE - Channel events file.
  WMQZVSE.MQFIEPE - Performance events file.
  WMQZVSE.MQFIEME - Command events file.
  WMQZVSE.MQFIENE - Configuration events file.

  The following files are sample definitions for user message queues:
  WMQZVSE.MQFI001
  WMQZVSE.MQFO001
  WMQZVSE.MQFI002
  WMQZVSE.MQFO002
  WMQZVSE.MQFI003
  WMQZVSE.MQFO003

You are strongly recommended to define one local queue in each physical file. If you intend to use the automatic VSAM reorganization feature with a queue, that queue must be the only queue in a physical VSAM file.

Installing security

You can protect your WebSphere MQ subsystem from unauthorized access by activating the WebSphere MQ for z/VSE security feature. For full details on the security feature, refer to Chapter 12, “Security,” on page 651.

Before installing security, ensure that your environment includes the following prerequisite systems:

- z/VSE 3.1 or above.
- CICS TS 1.1 or above.
- External Security Manager or Basic Security Manager (see below).

You must have an External Security Manager (ESM) that supports the SAF RACROUTE interface. WebSphere MQ for z/VSE is not dependent on any specific ESM; however, your ESM should recognize and support standard RACROUTE macro calls. For more information, contact your ESM vendor.

With z/VSE 4.3 or later, the Basic Security Manager (BSM) can now be used to secure WebSphere MQ for z/VSE.

If you have the correct prerequisites and intend to install WebSphere MQ for z/VSE security for your queue manager, you must copy and edit the SYSIN.Z installation file, available in the WebSphere MQ installation library PRD2.WMQZVSE. You must also change the MQJSETUP.Z sample JCL file that processes the SYSIN.Z file.

The SYSIN.Z file contains installation and configuration parameters that generally should not be changed. However, the file also contains switches for security, which are set off by default and need to be set on to activate security.

To activate the security feature, use the new SYSIN parameter SET to overwrite the default values for QM-SUBSYSID and QM-STATUS-SECURITY contained in the SYSIN.Z member in your installation sublibrary.

1. **QM-SUBSYSID**

   The default value for this parameter is MQV1. This parameter, the subsystem identifier (SSID), is used to build security resource names when performing security checks. You should set this parameter to a 4-character value that uniquely identifies your queue manager. If you set the SSID to blanks (spaces), the queue manager name is used to build resource names rather than the SSID. This is not recommended as this can lead to resource names that are too long for some ESMs. An SSID has to be set in order to use the Basic Security Manager.

2. **QM-STATUS-SECURITY**

   To activate security, set to ENABLED.

For example:

```
// DLBL LOADFL,'wmqzvse.mqfsset',,VSAM,CAT=?cat-name?
// EXEC IESVSMLD,SIZE=AUTO
80,E,LOADFL
SET QM-SUBSYSID VSE1
SET QM-STATUS-SECURITY ENABLED
* $$ SLI MEM=SYSIN.Z,S=prd2.wmqzvse
/*
```

or

```
// DLBL CONFIG,'wmqzvse.mqfcnfg',,VSAM,CAT=VSESPUC
// LIBDEF PHASE,SEARCH=(prd2.wmqzvse,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPUTIL,SIZE=MQPUTIL
UPDATE
SET QM-SUBSYSID VSE1
SET QM-STATUS-SECURITY ENABLED
 * $$ SLI MEM=SYSIN.Z,S=prd2.wmqzvse
/*
```

Once you have made these changes, you can run the MQJSETUP.Z sample JCL to import the contents of any SET parameters and the SYSIN.Z file into a VSAM ESDS. The ESDS is processed by installation transaction MQSU to build your

starting WebSphere MQ subsystem configuration. See “Starting WebSphere MQ” on page 27. Security installation is not complete until you run the MQSU transaction.

Changing the MQER TDQ definition

Security installation may also require changes to the MQER transient data queue (TDQ) definition of WebSphere MQ for z/VSE. The default definition for this TDQ is shipped in file MQCICDCT.A (see “Preparing CICS for WebSphere MQ” on page 25).

The MQER TDQ definition requires a trigger transaction to be fired every time an entry is written to the TDQ. The transaction that is started is also called MQER. With CICS TS, this transaction will run as the CICS default user (DFLTUSER) unless the DCT definition identifies a USERID.

For security purposes, the user identified with the MQER transaction must have WebSphere MQ CONNECT authority and UPDATE authority to the SYSTEM.LOG queue. Therefore, you must decide whether to grant these privileges to the CICS default user, or to a special user. For security purposes, we recommended that you identify a special user to run the MQER transaction.

If you intend to grant the appropriate authority to the CICS default user, you do not need to change the MQCICDCT.A sample file. However, if you intend to identify a special user to run the MQER transaction, you need to perform the following:

1. Create a user with your ESM.
2. Grant CONNECT and UPDATE authority to the user. For details on granting security access to users, refer to Chapter 12, “Security,” on page 651.
3. Copy the MQCICDCT.A file. We recommend that you copy the MQCICDCT.A file rather than directly edit the base file. The MQCICDCT.A file is a source fragment that should be included in the DCT source file for your CICS system.
4. Change the MQER TDQ definition in MQCICDCT.A as follows:

   Change:

```
MQER     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQER,
                DESTFAC=FILE,
                TRANSID=MQER,
                TRIGLEV=1
```

   To:

```
MQER     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQER,
                DESTFAC=FILE,
                USERID=youruser,
                TRANSID=MQER,
                TRIGLEV=1
```

5. Rebuild your DCT phase. Your CICS system programmer can use the MQCICDCT.A source fragment to do this.

Changing the MQXP TDQ definition

Similar to changes to the MQER TDQ definition, security installation may also require changes to the MQXP transient data queue (TDQ). The default definition for this TDQ is shipped in file MQCICDCT.A (see Preparing CICS for WebSphere MQ on page 14).

The MQXP TDQ is used by the WebSphere MQ queue manager to expire messages. To expire messages, the MQXP TDQ defines a trigger transaction that is started by CICS when an expiry request is written to the TDQ by the queue manager. The transaction that is started is also called MQXP. With CICS TS, this transaction will run as the CICS default user (DFLTUSER) unless the DCT definition identifies a USERID.

For security purposes, the user associated with the MQXP transaction must have WebSphere MQ CONNECT authority and UPDATE authority to the any ReplyToQ that might exist in the MQMD data structure of an expiring message. The user must also have UPDATE authority to any VSAM file that can contain expired messages. In other words, the MQXP transaction must be run by a user that has UPDATE authority to most, if not all, local queues.

For this reason, it is not recommended that the MQXP transaction runs with the authority of the CICS default user. Instead, it is recommended that the definition for the MQXP TDQ is changed to identify a USERID with the appropriate authority.

To change the MQXP TDQ definition:

1. Create a user with your ESM.
2. Grant CONNECT and queue UPDATE authority to the user. Also ensure that the user has UPDATE authority to relevant VSAM files. For more information about security access to users, refer to Chapter 12, “Security,” on page 651.
3. Copy the MQCICDCT.A file. We recommend that you copy the MQCICDCT.A file rather than directly edit the base file. The MQCICDCT.A file is a source fragment that should be included in the DCT source file for your CICS system.
4. Change the MQXP TDQ definition in MQCICDCT.A.

   Change:

   ```
   MQXP DFHDCT TYPE=INTRA,
               RSL=PUBLIC,
               DESTID=MQXP,
               DESTFAC=FILE,
               TRANSID=MQXP,
               TRIGLEV=1
   ```

   To:

   ```
   MQXP DFHDCT TYPE=INTRA,
               RSL=PUBLIC,
               DESTID=MQXP,
               DESTFAC=FILE,
               USERID=youruser,
               TRANSID=MQXP,
               TRIGLEV=1
   ```

5. Rebuild your DCT phase. Your CICS system programmer can use the MQCICDCT.A source fragment to do this.

Changing the MQIE TDQ definition

Similar to changes to the MQXP TDQ definition, security installation may also require changes to the MQIE transient data queue (TDQ). The default definition for this TDQ is shipped in file MQCICDCT.A (see "Preparing CICS for WebSphere MQ" on page 25).

The MQIE TDQ is used by the WebSphere MQ queue manager to register Instrumentation Event (IE) requests. An instrumentation event is a logical combination of conditions 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. To achieve this, the queue manager places an IE request on the MQIE transient data queue. Such requests are processed by the IE processor transaction, also called MQIE.

For security purposes, the user associated with the MQIE transaction must have WebSphere MQ CONNECT authority and UPDATE authority to the event queues. The event queues are identified by the queue manager's global system definition. The user must also have UPDATE authority to the VSAM files that host the event queues.

Rather than allowing the MQIE transaction to run as the CICS default user, it is recommended that the definition for the MQIE TDQ is changed to identify a USERID with the appropriate authority.

To change the MQIE TDQ definition:

1. Create a user with your ESM.
2. Grant CONNECT and queue UPDATE authority for each of the event queues to the user. Also ensure that the user has UPDATE authority to relevant VSAM files. For more information about security access to users, refer to Chapter 12, "Security," on page 651.
3. Copy the MQCICDCT.A file. It is recommended that you copy the MQCICDCT.A file rather than directly edit the base file. The MQCICDCT.A file is a source fragment that should be included in the DCT source file for your CICS system.
4. Change the MQIE TDQ definition in MQCICDCT.A. Change:

```
MQIE  DFHDCT  TYPE=INTRA,
      RSL=PUBLIC,
      DESTID=MQIE,
      DESTFAC=FILE,
      TRANSID=MQIE,
      TRIGLEV=1
```

   to:

```
MQIE  DFHDCT  TYPE=INTRA,
      RSL=PUBLIC,
      DESTID=MQIE,
      DESTFAC=FILE,
      USERID=youruser,
      TRANSID=MQIE,
      TRIGLEV=1
```

5. Rebuild your DCT phase. Your CICS system programmer can use the MQCICDCT.A source fragment to do this.

Changing the MQAC TDQ definition

Similar to changes to the MQIE TDQ definition, security installation may also require changes to the MQAC transient data queue (TDQ). The default definition for this TDQ is shipped in file MQCICDCT.A (see "Preparing CICS for WebSphere MQ" on page 25).

The MQAC TDQ is used by the WebSphere MQ queue manager to store accounting messages. Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications. To achieve this, the queue manager places an AC request on the MQAC transient data queue. Such requests are processed by the AC processor transaction, also called MQAC. For security purposes, the user associated with the MQAC transaction must have WebSphere MQ CONNECT authority and UPDATE authority to the system accounting queue. The accounting queue by default is SYSTEM.ADMIN.ACCOUNTING.QUEUE. The user must also have UPDATE authority to the VSAM file that hosts the accounting queue.

Rather than allowing the MQAC transaction to run as the CICS default user, it is recommended that the definition for the MQAC TDQ is changed to identify a USERID with the appropriate authority.

To change the MQIE TDQ definition:

1. Create a user with your ESM.
2. Grant CONNECT and queue UPDATE authority for the system accounting queue to the user. Also ensure that the user has UPDATE authority to relevant VSAM file. For more information about security access to users, refer to Chapter 12, "Security," on page 651.
3. Copy the MQCICDCT.A file. It is recommended that you copy the MQCICDCT.A file rather than directly edit the base file. The MQCICDCT.A file is a source fragment that should be included in the DCT source file for your CICS system.
4. Change the MQAC TDQ definition in MQCICDCT.A. Change:

```
MQAC DFHDCT TYPE=INTRA,
               RSL=PUBLIC,
               DESTID=MQAC,
               DESTFAC=FILE,
               TRANSID=MQAC,
               TRIGLEV=1
```

   to:

```
MQAC DFHDCT TYPE=INTRA,
                 RSL=PUBLIC,
                 DESTID=MQAC,
                 DESTFAC=FILE,
                 USERID=youruser,
                 TRANSID=MQAC,
                 TRIGLEV=1
```

5. Rebuild your DCT phase. Your CICS system programmer can use the MQCICDCT.A source fragment to do this.

Accounting data records written to the MQAC TDQ can be up to 32K in length. Consequently, the file associated with the MQAC TDQ defined to your CICS system must allow records up to this length.

For example, the default definition for the MQAC TDQ is TYPE=INTRA and DESTFAC=FILE. To accommodate TDQ records up to 32K, the DFHNTRA file might be defined as follows:

```
DEFINE CLUSTER (NAME(MQM300.CICSXX.DFHNTRA) -
                RECSZ(4089  32000) -
⋮
```

and your CICS startup JCL will identify this file in the DFHNTRA DLBL. For example:

```
// DLBL DFHNTRA,'MQM300.CICSXX.DFHNTRA',,VSAM,CAT=yourcat
```

Accounting and statistics messages can be up to 80k bytes in length. Consequently, when the system queues for these messages are defined you must specify a MAXMSGL of at least 80000.

Other considerations for installing security

Other installation steps involve:

1. Activation of security classes.
2. Creation of ESM resources.
3. Creation of users.
4. Assignment of resource permissions to users.

Each of these is covered in detail in Chapter 12, “Security,” on page 651.

Preparing CICS for WebSphere MQ

Various CICS tables and definitions must be created and customized for use by the WebSphere MQ subsystem.

You must define the following:

- CICS resources into the CICS CSD.
- Entry definitions for the CICS Destination Control Table.
- Entry definitions for the CICS File Control Table.

The definitions should be reviewed by your CICS systems programmer.

Use the samples (see Appendix D, “Sample JCL and programs,” on page 977) provided with the product. See Appendix A, “CICS control table definitions,” on page 711 for further information.

To help you install the PCT and PPT CICS definitions, the sample MQJCSD.Z is provided. MQJCSD.Z automatically defines the WebSphere MQ entries required into the CICS Definition Data Set (without using migrated CICS, DFHPPT and DFHPCT tables). If you are installing WebSphere MQ for z/VSE in a CICS TS environment, you should use the MQJCSD24.Z sample rather than the MQJCSD.Z sample.

You may need to modify this sample to fit your own environment, because all entries are defined in group “MQM”, which is then added to the VSELIST list.

MQJCSD.Z - Define CICS resources for CICS for z/VSE
: Sample code that can be used to create CICS-specific PCT and PPT definitions, which are required by the WebSphere MQ subsystem.

MQJCSD24.Z - Define CICS resources for CICS TS
: Sample JCL that can be used to create PCT, PPT, and FCT definitions specific to CICS TS that are required by the WebSphere MQ subsystem.

MQCICFCT.A - File Control Table (FCT)

The sample code provided can be used for creating CICS definitions for the WebSphere MQ configuration and sample queue files. These definitions may require changing to your site's specific requirements.

Note: If you install under CICS TS, you do not need to create your File Control Table (FCT) definitions with this sample. File definitions are provided in the MQJCSD24.Z sample JCL file. It should also be noted that WebSphere MQ files cannot be defined as remote to your CICS system.

MQCICDCT.A - Destination Control Table (DCT)

The WebSphere MQ product requires intrapartition transient data queues (TDQ) MQER, MQXP and MQIE, for the processing of log, message expiry and instrumentation events respectively

Note: If you install the security feature, you may need to make special changes to the MQER, MQXP and MQIE transient data queue definitions. See "Installing security" on page 19 for more details.

Modify CICS start-up deck

For CICS applications to use the WebSphere MQ facilities, you must inform CICS of the WebSphere MQ configuration and workfiles, and the location of the WebSphere MQ for z/VSE phases as follows:

- Add the label definitions for the CICS start-up job (MQJLABEL.Z) to your CICS start-up deck, or to the standard label procedures. It contains information about the datasets that WebSphere MQ for z/VSE uses.

  This step is not necessary when WebSphere MQ is running in a CICS TS environment. However, if required, label definitions in the CICS TS startup JCL can be used to override MQ VSAM file definitions in the CSD.

  This file **must** be modified and customized to refer to the correct volume identifiers and catalog names.

  This should be done by your z/VSE systems programmer.
- Add the WebSphere MQ for z/VSE subsystem install library defined in "Installing WebSphere MQ for z/VSE - all users" on page 16 (default name "PRD2.WMQZVSE") to the LIBDEF control statement in your CICS startup deck.
- If you are using TCP/IP for queue manager to queue manager or client connections, you must also ensure the PRD1.BASE (TCP/IP base library) is concatenated ahead of the PRD2.SCEEBASE (LE base library). This will ensure that the TCP/IP runtime is correctly referenced.

  For example:

```
// LIBDEF       *,SEARCH=(PRD2.WMQZVSE,  *
                      PRD2.CONFIG,        *
                      PRD1.BASE,          *
                      PRD2.SCEEBASE,      *
                      ...)
```

Recovery and restart

Although WebSphere MQ uses its own recovery and restart logic, it also uses standard CICS file management. When WebSphere MQ is running in a CICS for z/VSE environment, it is important that all MQ VSAM clusters are defined in the DFHFCT with the LOG parameter set to `YES`. In addition, the CICS logging facility should be activated with `JCT = xx` or `YES` in the DFHSIT.

When WebSphere MQ is running in a CICS TS environment, CSD file definitions for MQ datasets should be defined with RECOVERY(BACKOUTONLY).

If you do not fulfill the above conditions, unpredictable results can occur, such as loss of messages or inaccurate values for message sequence numbers.

CICS journal control table

The CICS journal control table (JCT) can be affected by the queue definitions. If a physical record is larger than the buffer size specified in the JCT, a CICS task abend of "AFCL" occurs.

The provided sample FCT queue definitions specify a maximum record length of 4089 bytes. If large records are written, you should set the BUFSIZE parameter of the CICS DFHJCT to a different value; a BUFSIZE value of 4200 is usually sufficient.

For further information, see the *CICS for z/VSE Resource Definition (Macro)* manual.

This is reflected in either the WebSphere MQ System Log or the CSMT TD queue when an MQPUT call is processed trying to perform this function.

Uppercase translation

Queue manager, queue and channel names are case sensitive on WebSphere MQ systems. If WebSphere MQ for z/VSE sends messages to other WebSphere MQ systems, you must specify UCTRAN = TRANID or UCTRAN = NO in your CICS terminal definitions.

If you do not do this, the names you enter into the WebSphere MQ panels are translated into uppercase, and they may not match the actual names on the target WebSphere MQ system.

Installation checkpoint (CICS)

You have now set up the CICS system, and it is ready to be restarted to update the system configuration and utilize the WebSphere MQ subsystem.

Note: If the CICS system has not been updated correctly, check through the preceding instructions to ensure that they all completed correctly.

Starting WebSphere MQ

The MQ CICS environment requires a cold start. Following the restart, the WebSphere MQ for z/VSE configuration file must be initialized and populated before the WebSphere MQ for z/VSE subsystem can be used.

You do this with the `MQSU` transaction. However, you are strongly recommended to ensure that all the WebSphere MQ for z/VSE subsystem files are available for access by CICS before running this job.

You do this by issuing the CICS transaction:

```
CEMT  INQUIRE FILE(MQF*)
```

All of the WebSphere MQ for z/VSE files defined in "Allocate and initialize the required WebSphere MQ files" on page 18 should be visible, and you should be able to open, close, enable, and disable the files.

If you cannot access these files, refer to your CICS systems programmer and review the steps in "Preparing CICS for WebSphere MQ" on page 25.

If the files are accessible, issue the transaction MQSU or MQSU UC for uppercase output from the WebSphere MQ for z/VSE administrator transactions. This completes with the message "`WMQ Install Completed, nnnn input records read`" or "`WMQ (UPPERCASE) INSTALL COMPLETED, nnnn INPUT RECORDS READ`", depending whether uppercase output was selected. The number of input records read may change depending on the current maintenance level.

Note that you may need to run the MQJSETUP job and the `MQSU` transaction after applying maintenance. Please check PTF cover letters.

WebSphere MQ initialization

Before you initialize your WebSphere MQ for z/VSE system, if you decide to install the security feature, you must carry out a basic security implementation first. For details of how to implement security, refer to Chapter 12, "Security," on page 651.

If you have already implemented security, or you do not wish to install the security feature, you can now initialize your WebSphere MQ for z/VSE subsystem as follows:

1. Set up the WebSphere MQ for z/VSE environment.

   Run `MQSE` (Setup Environment).

   The response "`MQSE:WMQ environment setup completed`" is displayed, after a few seconds.

2. Specify the queue manager name.

   There can be only one queue manager on each WebSphere MQ for z/VSE system and each WebSphere MQ system should have a unique queue manager name. The name is specified using the MQMT System Administration transaction, as follows:

   a. Enter the transaction code `MQMT` on a CICS terminal.
   b. Select option 1 for the "Configuration" menu.
   c. Select option 1 for the "Global System Definition" update screen.

```
2013/01/25        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
05:46:06                  Global System Definition                   CIC1
MQWMSYS                 Queue Manager Information                    A002
Queue Manager . . . . . .: PTHVSEC
Description Line 1. . . . .: ZVSE 4.3 SYSTEM ON PTHVSEC
Description Line 2. . . . .: ...
                          System Values
Maximum Connection Handles.: 00000100       System Wait Interval : 00000030
Maximum Concurrent Queues .: 00000100       Max. Recovery Tasks  : 0000
Allow TDQ Write on Errors  : Y   CSMT       Local Code Page  . . : 01047
Allow Internal Dump  . . . : Y              Subsystem id . . . . : MQV1
                                            Channel Auth Enabled : Y
                       Queue Maximum Values
Maximum Q Depth . . . . . .: 00010000       Maximum Global Locks.: 00001000
Maximum Message Size. . . .: 00409600       Maximum Local Locks .: 00001000
Maximum Single Q Access . .: 00000100       Max Properties Length: 00004094
                       Global QUEUE /File Names
Configuration File. : MQFCNFG
LOG Queue Name. . . : SYSTEM.LOG
Dead Letter Name. . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name. : SYSTEM.MONITOR
Requested record displayed.
PF2=Return PF3=Quit PF4/ENTER=Refresh    PF6=Update
                    PF9=Communications  PF10=Log    PF11=Events  PF12=Exits
```

Figure 5. Default global system definition

d. Change the "Queue Manager" field to the name that you are giving to your local queue manager.

e. Press function key six (PF6) to update the configuration.

f. Press function key three (PF3) to quit the screen.

You can leave the other fields unchanged.

3. Define system queues

The System Log is a local queue used to record system diagnostic and error messages. It should be defined before the MQ system is started for the first time.

The name of the system log queue is specified in the queue manager's global system definition. The default name for the queue is SYSTEM.LOG, however, this can be changed using MQMT option 1.1.

To create the system log queue, carry out the following:

a. Type MQMT at the system prompt, or use MQMB and go to Step 3d.

b. Type 1 on the main menu to select Configuration.

c. Type 2 on the Configuration menu to select queue definitions.

The "Queue Main Options" screen is displayed.

d. Complete the following fields:

- Object Type L
- Object Name SYSTEM.LOG

e. Press PF5 (Add) to display the "Local Queue Definition" screen.

f. Press PF5 (Add) to display the "Queue Extended Definition" screen and change the default values in the following fields:

- Usage to N (Normal)
- File name to MQFLOG
- Max. Q depth to 5000
- Max. msg length to 2048

g. Press PF5 (Add) to save the changes.

Once the system log queue has been defined to the queue manager the MQ system can be started. Although it is not immediately necessary, it is recommended that the system admin command and system admin reply queues are also defined at this time, repeating steps 3a on page 29 through 3g, but use file name MQFACMD and MQFARPY respectively to host these queues, and a maximum message length of 16000. It is also recommended that the system dead letter and system monitor queues are defined unless you plan to use the sample object definition job, MQJINSG, which will define these queues for you. For more details, see "Default object definitions" on page 37. The names of these queues are configurable using MQMT option 1.1. The default names for these queues, and their default CICS filenames are:

```
SYSTEM.DEAD.LETTER.QUEUE    MQFERR
SYSTEM.MONITOR              MQFMON
SYSTEM.ADMIN.COMMAND.QUEUE  MQFACMD
SYSTEM.ADMIN.REPLY.QUEUE    MQFARPY
```

If you are planning to use the WebSphere MQ Explorer to remotely administer your z/VSE queue manager, you should also define the system default queues. The default names for these queues, and their default CICS filenames are:

```
SYSTEM.DEFAULT.ALIAS.QUEUE  n/a
SYSTEM.DEFAULT.LOCAL.QUEUE  MQFDEFS
SYSTEM.DEFAULT.MODEL.QUEUE  MQFDEFS
SYSTEM.DEFAULT.REMOTE.QUEUE n/a
SYSTEM.MQEXPLORER.REPLY.MODEL MQFADMN
```

Once again, you can use the MQJINSG.Z sample MQSC job to create these queues, but you must define the system command and reply queues (explained above) before you can run the job. For more details, see "Default object definitions" on page 37.

In addition, if instrumentation events are required, it is recommended that the event queues are defined at this time. The event queues are specified as part of the queue manager's global system definition (MQMT 1.1, PF11). Sample VSAM files for the event queues are provided in file MQJQUEUE.Z, and file definitions for CICS are provided in MQCICFCT for CICS 2.3, and MQJCSD24.Z for CICS TS. The default names for these queues, and their default CICS filenames are:

```
SYSTEM.ADMIN.QMGR.EVENT           MQFIEQE
SYSTEM.ADMIN.CHANNEL.EVENT        MQFIECE
SYSTEM.ADMIN.PERFM.EVENT          MQFIEPE
SYSTEM.ADMIN.COMMAND.EVENT        MQFIEME
SYSTEM.ADMIN.CONFIGURATION.EVENT  MQFIENE
```

Lastly, if accounting and statistics messages are required, it is recommended that the relevant system queues are defined at this time. The names of these queues are not configurable. Sample VSAM files for the queues are provided in file MQJQUEUE.Z, and file definitions for CICS are provided in MQCICFCT for CICS 2.3, and MQJCSD24.Z for CICS TS. The default names for these queues, and their default CICS filenames are:

```
SYSTEM.ADMIN.ACCOUNTING.QUEUE MQFACCTS
SYSTEM.ADMIN.STATISTICS.QUEUE MQFSTATS
```

4. Define a local queue.

   You must define some local queues to test the operation of the WebSphere MQ for z/VSE subsystem. This task is also carried out by using the MQMT transaction.

   The following definitions allow the installation verification program, TST2, to send messages to `ANYQ`.

Carry out the following procedure:

a. Type MQMT at the system prompt.
b. Type 1 on the main menu to select Configuration.
c. Type 2 on the Configuration menu to select queue definitions. The "Queue Main Options" screen appears.
d. Complete the following fields:
   - Object Type L
   - Object Name ANYQ
e. Press PF5 (Add) to display the "Local Queue Definition" screen.
f. Press PF5 (Add) to display the "Queue Extended Definition"screen and change the default values in the following fields:
   - Usage mode N (Normal)
   - Physical File Name MQFI001 (file name from FCT)
   - Maximum Q Depth 00000100
   - Maximum Message Length 00002048
g. Press PF5 (Add) to save the changes.
h. Press PF2 (Options) to return to the Queue Main Options Screen.
i. Press PF9 (List) to display a selection screen.
j. On the selection screen, use the cursor keys to select the queue. Press any character key followed by the Enter key.

   A screen displays the queue parameters that you have entered. Check that the correct data has been entered.

5. Initialize the WebSphere MQ for z/VSE queue manager. There are two ways of doing this. Either:
   a. Type MQIT on a CICS terminal.

      The response "MQIT: No channel definitions. Initialization completed" is displayed when the process has completed. This is normal if you have not yet created any channels. Default channel definitions exist if you have run the MQJINSG.Z sample MQSC job.

      or

   b. Use the WebSphere MQ for z/VSE System Administration transaction (MQMT), as follows:
      1) Issue MQMT to display the main menu panel of WebSphere MQ Administration.
      2) Select 2 - Operation.
      3) Select 4 - Initialization/Shutdown.
      4) Type I in the function field and press function key six (PF6).

      Note: If you carry out the initialization before you perform system setup, you receive the message MQ900000:WMQ z/VSE ENVIRONMENT NOT INITIALIZED.

In the future, you can combine Step 1 on page 28 and Step 5 by issuing MQSE with the parameter I to perform the initialization step, as follows:

```
MQSE  I
```

The response "MQSE:WMQ environment setup and initialized" is displayed when the process has completed.

Checking MQ is active

When you have completed the steps listed in "WebSphere MQ initialization" on page 28, the queue manager is active and you can verify this by typing MQMT on a

CICS console, to display Figure 6.

```
12/09/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
14:45:45           *** Master Terminal Main Menu ***                CIC1
MQWMTP                                                              A001
                         SYSTEM IS ACTIVE

                           1. Configuration

                           2. Operations

                           3. Monitoring

                           4. Browse Queue Records


                    Option:




Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Clear/PF3=Exit                           Enter=Select
```

Figure 6. Master terminal main menu

Figure 6 shows the MQ Master Terminal main menu. Ensure that the "SYSTEM IS ACTIVE" message is displayed.

WebSphere MQ installation verification test

The WebSphere MQ subsystem is now ready for the installation verification procedures.

Stop the WebSphere MQ subsystem, using either the MQST transaction, or the Operations Shutdown menu – MQMT option 2.4, and then reinitialize the WebSphere MQ subsystem (see "WebSphere MQ initialization" on page 28).

To carry out the installation verification test you need:

- One local queue.
- The sample transaction TST2.
- The program TTPTST2 provided with the product.
- Access to two terminals.

Local queue verification test

The local queue verification test consists of five steps:

1. Initialize the WebSphere MQ runtime environment.
2. Use the test program TTPTST2 to send a number of messages.
3. Use MQMT to verify that these messages are on the queue.
4. Use the test program TTPTST2 to read the messages.
5. Use MQMT to verify that the messages have been delivered.

Step 1 (initializing the WebSphere MQ runtime environment) is achieved by running transaction MQSE, and either MQIT or MQMT option 2.4. Steps 2 through 5 are achieved as follows:

1. On one terminal, issue the transaction code TST2. This invokes the WebSphere MQ for z/VSE test program TTPTST2 and produces the screen in Figure 7.

```
TST2 is a test facility for SENDING / RECEIVING messages
The format of command is as follows:
TST2 XXXX NN QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ

(NOTE: parameters are separated by space(s)).
 XXXX  4-character function code, pad with trailing blank
                     HELP - DISPLAY THIS HELP TEXT
                     PUT  - MQPUT  MESSAGES
                     PUT1 - MQPUT1 MESSAGES
                     PUTR - MQPUT W/ REPLY MESSAGE
                     GET  - MQGET  MESSAGES
                     GETD - MQGET W/ BROWSE & DELETE
                     BOTH - MQPUT FOLLOWED BY MQGET
                     INQ  - INQ ABOUT QUEUE (no count NN)
 NN    2-digit number with leading zero (01 TO 99)
 QQQQ  A 48-character field giving the name of a queue.
 An additional prompt will ask for the name of the reply queue for PUTR option.
```

Figure 7. TTPTST2 screen

2. On a second terminal, start MQMT and use option 3.1 to monitor queue operations. This displays the screen in Figure 8.

```
12/09/2008           IBM WebSphere MQ for z/VSE Version 3.0.0        TSMQ300
15:02:45                   Monitor Queues                            CIC1
MQWMMOQ                                                              A001
                           QUEUING SYSTEM IS ACTIVE
S QUEUE                         FILE     T INBOUND  OUTBOUND        LR   QDepth
  ANYQ                          MQFI001  N IDLE     IDLE             0        0
  SYSTEM.DEAD.LETTER.QUEUE      MQFERR   N IDLE     IDLE             0        0
  SYSTEM.LOG                    MQFLOG   N IDLE     IDLE             0        4
  SYSTEM.MONITOR                MQFMON   N IDLE     IDLE             0        0









Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit  PF7=Back  PF8=Forward
                              PF9=All   Select or PF10=Detail
```

Figure 8. Monitor queues screen

3. On the first terminal, issue:

```
TST2 PUT 10 ANYQ
```

Note: If you type TST2 without parameters, the HELP screen for using TTPTST2 is displayed.

4. TTPTST2 sends the specified messages addressed to ANYQ.

   You receive the following message on successful completion of the transaction:

   ```
   FULL CYCLE HAS BEEN PERFORMED SUCCESSFULLY
     QUEUE USED - ANYQ
     NUMBER OF MESSAGES PROCESSED -  10
     TOTAL SECONDS ..................... -  hh:mm:ss
   ```

   where:
   - 10 is the number of messages you specified (nn).
   - hh:mm:ss is the time taken to process nn messages.
5. Return to the terminal running the MQMT Monitor Queue process.
6. Press the Enter key on this terminal.

   The QDEPTH column for queue ANYQ now equals 10. This is the value specified for nn in Step 4.
7. Messages on an WebSphere MQ for z/VSE queue can be displayed at any time using the MQMT Browse Queue facility (MQMT option 4). Select this option, enter the queue name in the "Object" field, and press the Enter key.

   This displays the screen in Figure 9.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:05:20                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A001

    Object Name: ANYQ
    QSN Number : 00000001       LR-        0, LW-       10, DD-MQFI001
                      Queue Data Record
Record Status : Written.        PUT date/time  : 20061013093221
Message Size  : 00000200        GET date/time  :
Offset ....+....!....+....!....+....!....+....!....+....!....+....!....+....!
 00000 THIS IS A MESSAGE TEXT
 00070
 00140









Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Hex       PF10=Hdr   PF11=MD
```

Figure 9. Browse Queue Records screen - status written

   The queue can then be browsed forwards and backwards using function keys four and five (PF4 and PF5).

   Note that in this example, the "Record Status" field is Written. This indicates that the message has been placed on the queue but not retrieved.
8. Move to the other terminal.
9. At the CICS prompt, type:

   ```
   TST2 GETD  10 ANYQ
   ```

   Note: If you type TST2 without parameters, the HELP screen for using TTPTST2 is displayed.
10. TTPTST2 reads the specified messages from ANYQ.

You receive the following message on successful completion of the transaction:

```
FULL CYCLE HAS BEEN PERFORMED SUCCESSFULLY
  QUEUE USED - ANYQ
  NUMBER OF MESSAGES PROCESSED -  10
  TOTAL SECONDS ..................... -  hh:mm:ss
```

where:

- `10` is the number of messages you specified (`nn`).
- `hh:mm:ss` is the time taken to process `nn` messages.

11. Return to the terminal, running the MQMT Monitor Queue process.
12. Press the `Enter` key. The Monitor Queue screen still displays ANYQ as the only defined non-system queue.

    Note:

    a. The QDEPTH number, representing the number of messages on the queue, has decreased to zero.

    b. The total number of messages read from the queue (LR) has increased by the number you read using TTPTST2.

13. Use the MQMT Browse facility to view ANYQ. The "Record Status" field has changed to `Deleted`, and the "GET date/time" field is now completed.

    This indicates that the record has now been retrieved by an application. In this case the test transaction TST2 was used with parameters "TST2 GETD 5 ANYQ".

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:05:20                   Browse Queue Records                      CIC1
MQWDISP                      SYSTEM IS ACTIVE                        A001

    Object Name: ANYQ
    QSN Number : 00000001       LR-        5, LW-       10, DD-MQFI001
                      Queue Data Record
Record Status : Deleted         PUT date/time  : 20061013093221
Message Size  : 00000200        GET date/time  : 20061013094517
Offset ....+....!....+....!....+....!....+....!....+....!....+....!....+....!
 00000 THIS IS A MESSAGE TEXT
 00070
 00140









Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Hex       PF10=Hdr   PF11=MD
```

Figure 10. Browse Queue Records screen - status deleted

Note that WebSphere MQ for z/VSE differs from many other WebSphere MQ platforms, in that when a message is retrieved from a queue it is logically deleted but not physically deleted. The messages are merely flagged as "Deleted".

As a consequence of this technique of flagging messages as "written" and "deleted", messages can have their logical state changed, and where necessary, reprocessed.

You can do this using MQMT option 2.5. However, you are advised to carry out this procedure only when you are familiar with the WebSphere MQ for z/VSE system.

You have now completed a local installation verification test demonstrating that two applications can send and receive messages through an WebSphere MQ queue.

Installation checkpoint (installation verification test)

You can now:

- Define local queues.
- Start and stop the queue manager.
- Browse queues using MQMT.
- Monitor the status of queues.
- Run simple WebSphere MQ programs that use local queues.

Note: If the installation verification test has not completed, check through the preceding instructions.

Remote queue verification test

In order to expand this test to include a remote link, you must carry out the following steps:

1. Using the appropriate manufacturer's directions, install the prerequisite hardware and software required to support the selected transport protocol (SNA LU 6.2 or TCP/IP).
2. Define the WebSphere MQ channels that you require. See "Channel definitions" on page 112, and coordinate this task with the remote system administrator.

   You will need to define a Sender channel to send messages from WebSphere MQ for z/VSE to a remote MQ system, and a Receiver channel to receiver message from a remote MQ system.

   Sender and receiver channels operate in pairs. They must have the same name, for example, a Sender channel under WebSphere MQ might be called VSE1.TO.NT5, and the Receiver channel on the remote NT system would also be called VSE1.TO.NT5. Sender and Receiver channel pairs must also have matching channel parameter values, that is, matching maximum messages size, batch size and wrap sequence.
3. Configure the transmission queues and remote queues required by WebSphere MQ to communicate over the channel – see "Queue definitions" on page 96.

   For the remote queue verification test, you will need to define a remote queue that identifies a local queue on a remote queue manager, and a transmission queue used to temporarily store messages while they are transmitted to the remote queue manager.

   To test remote queuing to WebSphere MQ for z/VSE, you will also need a local queue that can be identified in a remote queue definition on a remote queue manager.

Use transaction TST2 to place test messages on a remote queue. These messages will be temporarily stored in the transmission queue identified in the remote queue definition. The WebSphere MQ for z/VSE queue manager will subsequently transmit the message to the remote queue manager. Verify that the test messages arrive successfully on the remote system.

Remote MQ systems will have a utility program similar to the TST2 transaction. On some systems, this program is called 'amqsput'. Use the utility program on the

remote system to put messages on the remote queue that points to a local queue on the WebSphere MQ for z/VSE system. Verify that the test messages arrive successfully on the local queue.

You have now installed and locally verified WebSphere MQ and you can use the administrative programs and the MQI libraries.

However, before your user applications can effectively use the system for message transmission, you must fully configure the system with your queue definitions.

This last step is the most important part of the installation. The requirements are detailed in:

- Chapter 3, "Configuring network communications," on page 43, which provides the configuration guidelines.
- Chapter 4, "System operation," on page 77, which describes the WebSphere MQ system administration screens used in the configuration.

Default object definitions

The sample file MQJINSG.Z provides sample JCL to create default objects to your queue manager. This job creates the following objects:

Queues

SYSTEM.ADMIN.ACCOUNTING.QUEUE
System accounting queue
SYSTEM.ADMIN.CHANNEL.EVENT
Channel event queue
SYSTEM.ADMIN.PERFM.EVENT
Performance event queue
SYSTEM.ADMIN.QMGR.EVENT
Queue manager event queue
SYSTEM.ADMIN.STATISTICS.QUEUE
System statistics queue
SYSTEM.CICS.BRIDGE.QUEUE
WMQ-CICS bridge queue
SYSTEM.DEAD.LETTER.QUEUE
Dead letter queue
SYSTEM.DEFAULT.ALIAS.QUEUE
Default alias queue
SYSTEM.DEFAULT.LOCAL.QUEUE
Default local queue
SYSTEM.DEFAULT.MODEL.QUEUE
Default model queue
SYSTEM.DEFAULT.REMOTE.QUEUE
Default remote queue
SYSTEM.MONITOR
MQI monitor queue
SYSTEM.MQEXPLORER.REPLY.MODEL
MQ Explorer model reply queue

Channels

SYSTEM.ADMIN.SVRCONN
MQ Explorer svrconn channel
SYSTEM.AUTO.RECEIVER
Receiver for Channel auto-def
SYSTEM.AUTO.SVRCONN
SvrConn for Channel auto-def

SYSTEM.DEF.RECEIVER
Default receiver channel
SYSTEM.DEF.REQUESTER
Default requester channel
SYSTEM.DEF.SENDER
Default sender channel
SYSTEM.DEF.SERVER
Default server channel
SYSTEM.DEF.SVRCONN
Default server connection channel
SYSTEM.AUTO.RECEIVER
Channel auto-definition receiver channel
SYSTEM.AUTO.SVRCONN
Channel auto-definition server connection channel

Namelists
SYSTEM.DEFAULT.NAMELIST
Default namelist object

Listeners
SYSTEM.DEFAULT.LISTENER.TCP
Default TCP/IP listener object

The MQJINSG.Z sample job uses the WebSphere MQ Command (MQSC) utility which uses the WebSphere MQ for z/VSE Batch Interface. Consequently, to run the sample job, the VSE queue manager and batch interface must be active. In addition, the MQSC utility uses the system admin command and system admin reply queues. These must be defined to the queue manager before the sample can be run. Like other samples, you must copy and edit the MQJINSG.Z file to customize the JCL.

Note: You can use the MQJINSG.Z sample as the basis for creating your own jobs to define, alter, or delete queues and channels.

Post installation verification test CICS modifications

The WebSphere MQ for z/VSE subsystem can be started and stopped automatically as part of the normal CICS startup and shutdown procedures. You do this by adding appropriate entries to the CICS Initialization and Shutdown parameters.

You **must** not carry out these steps until you have installed WebSphere MQ for z/VSE.

CICS Program List Table Post Initialization (PLTPI)
The WebSphere MQ subsystem requires initialization before applications can start using the queue manager. These steps set up the WebSphere MQ environment and initialize the WebSphere MQ resources.

To start WebSphere MQ automatically, you can add the following programs to the CICS initialization PLT (PLTPI) list:
MQPSENV
Set up the WebSphere MQ environment.
MQPSTART
Initialize the resources.

For example:

```
DFHPLT TYPE=ENTRY, PROGRAM=MQPSENV DFHPLT TYPE=ENTRY, PROGRAM=MQPSTART
```

Other methods are given in “WebSphere MQ initialization” on page 28.

CICS Program List Table Shut Down (PLTSD)

The WebSphere MQ subsystem should be shutdown correctly before shutting down CICS. This can be done:

- Manually, using transaction MQST.
- Automatically, by placing the WebSphere MQ program MQPSTOP in the CICS shutdown PLT before the DFHDELIM statement.

  For example:

  ```
  DFHPLT TYPE=ENTRY, PROGRAM=MQPSTOP
  ```

  This ensures that WebSphere MQ ends during the first phase of CICS shutdown.

Migration procedures for existing users

Please review the section “Installing WebSphere MQ for z/VSE - all users” on page 16 before proceeding with the instructions in this section.

If you are migrating from MQSeries for VSE V1.4, you must run the sample jobs MQJMIGR1 and MQJMIGR2 before proceeding with the following steps. This samples can be found in the WebSphere MQ for z/VSE installation sublibrary.

Conveniently, migration from MQSeries for VSE to WebSphere MQ for z/VSE does not require the deletion and re-creation of your VSAM datasets. This means your existing files can be used with their existing data. The only exception to this is the WMQ configuration file.

To carry out the migration, follow the procedure under “Installing WebSphere MQ for z/VSE - all users” on page 16 with the following modifications:

1. Do not run sample job MQJCONFG.Z.

   Running MCJCONFG.Z deletes and redefines your WMQ configuration file, which contains, among other things, your application queue definitions.

   Instead, the execution of the MQJSETUP.Z job and transaction MQSU ensures your configuration is upgraded correctly for V3. MQJSETUP.Z and MQSU are standard steps during installation.
2. Do not run sample job MQJQUEUE.Z.

   The MQJQUEUE.Z job deletes and redefines your WMQ VSAM datasets. You must not run this job if you want to keep your existing queue data.
3. Check SENDER channel definitions.

   If you are migrating from V1.4 or V2.1.0 and conversion of message data is not to be done by the sender MCA, then set the **Convert msgs(Y/N)** field to N for all sender channel definitions.

   Note: To keep existing V1.4 and V2.1.0 behaviour of the sender channel, you should set the Convert msgs(Y/N) field to N in all sender channel definitions.
4. When migrating from V1.4 or V2.1.0, you must now relink all your WMQ applications with LIBDEF pointing to the V3 installation sublib in order to include the new WMQ for z/VSE application programming interface objects. Relinking is required for CICS and batch applications.
5. Correct application design in WebSphere MQ for z/VSE requires that the MQI calls be done by the program issuing the MQCONN or by programs EXEC CICS LINKed from this program.

   For example:

```
PROG1  MQCONN
   MQOPEN
   MQGET
   CICS SYNCPOINT
   MQCLOSE
   MQDISC
   EXEC CICS RETURN
```

or

```
PROG1  EXEC CICS LINK PROG2

   PROG2  MQCONN
      MQOPEN
      MQGET
      CICS SYNCPOINT
      MQCLOSE
      MQDISC
      EXEC CICS RETURN
```

or

```
PROG1  EXEC CICS LINK PROG2

   PROG2  MQCONN
      EXEC CICS LINK PROG3

      PROG 3  MQOPEN
         MQGET
         CICS SYNCPOINT
         MQCLOSE
         EXEC CICS RETURN
      MQDISC
      EXEC CICS RETURN
   EXEC CICS RETURN
```

In MQSeries for VSE, an application could EXEC CICS LINK to a program to perform an MQCONN and then EXEC CICS RETURN and use the connection handle to perform other MQI calls in the calling program or perform EXEC CICS LINKed to other programs.

To allow the same behavior, the installation compatibility mode can be set in the configuration file by specifying SET QM-COMPAT-MODE ENABLED as the statement preceding the SYSIN.Z statements when updating the configuration file.

For example:

```
// DLBL LOADFL,'wmqzvse.mqfsset',,VSAM,CAT=?cat-name?
// EXEC IESVSMLD,SIZE=AUTO
80,E,LOADFL
SET QM-COMPAT-MODE  ENABLED
* $$ SLI MEM=SYSIN.Z,S=prd2.wmqzvse
/*
```

or

```
// DLBL CONFIG,'wmqzvse.mqfcnfg',,VSAM,CAT=VSESPUC
// LIBDEF PHASE,SEARCH=(prd2.wmqzvse,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPUTIL,SIZE=MQPUTIL
UPDATE
SET QM-COMPAT-MODE  ENABLED
* $$ SLI MEM=SYSIN.Z,S=prd2.wmqzvse
/*
```

When, after completing the full installation process with the above modifications, you start WMQ for z/VSE 3.0, you should see your existing queues with their previous data, and your existing channel definitions. This completes the migration process for V1.4 and V2 systems.

Note: If the migration has not completed correctly, check through the preceding instructions.

Chapter 3. Configuring network communications

This chapter describes the steps you perform to configure WebSphere MQ to run on the CICS system and communicate with other WebSphere MQ systems. The chapter assumes that your chosen communications software has been installed and correctly configured on your system.

For ACF/VTAM, using WebSphere MQ should not require any changes to the:
- VTAM parameters.
- Definition of CICS systems to VTAM.

However, you must define all the LUs that are involved.

For TCP/IP, using WebSphere MQ with the TCP/IP communications protocol requires the installation of TCP/IP for z/VSE V1.5F (or equivalent) or later.

TCP/IP is shipped as part of the z/VSE base product in library PRD1.BASE, and simply requires that you install a product key together with your customer information. For further details refer to the *TCP/IP for z/VSE User's Guide*.

WebSphere MQ for z/VSE does not have any special TCP/IP installation or configuration requirements.

Note: If TCP/IP is to be used as a transport protocol, the TCP/IP phase sublibrary must be added to the LIBDEF statement in the CICS startup JCL before the SCEEBASE sublibrary.

This is because SCEEBASE contains a TCP/IP phase stub that handles TCP/IP API calls when TCP/IP is not installed.

This chapter describes how to define connections and sessions for LU 6.2 channel connections, and provides guidelines for configuring the queue manager, channels and queues for effective communications.

WebSphere MQ system definitions required for ACF/VTAM

The local WebSphere MQ for z/VSE system has to be informed about remote WebSphere MQ systems with which it will communicate. WebSphere MQ has to be defined to:

- WebSphere MQ on CICS (in the network specific parts of the channel definition)
- CICS itself, in one of these ways:
  - In a TERMINAL definition.
  - In CONNECTION/SESSION definitions.
  - Through the CICS AUTOINSTALL facility.
- VTAM (either predefined, or by VTAM dynamic resource definition), if you are using SNA LU6.2.

Definitions in CICS for LU 6.2 connections

If the CICS end of an WebSphere MQ channel is to initiate the channel connection (that is, the CICS channel-endpoint is a sender), CICS performs an EXEC CICS ALLOCATE. However, this succeeds only if CICS is:

- A contention winner.
- Already bound.
- Not already allocated.

If CICS has no definition of the resource, CICS is incapable of formulating a request to VTAM for session establishment. In these circumstances, CICS AUTOINSTALL is inappropriate - autoinstall is for incoming session establishment requests, not for outgoing ones.

Therefore, for sender channel-endpoints on z/VSE, a definition of the remote system is required at the CICS level.

If the remote system, at the network level, is capable of supporting parallel sessions (for example, it has independent LU 6.2 capability, or it is another CICS system) and, you intend to configure several channels between the two systems, you should use CONNECTION and SESSIONS definitions.

Typical definitions, using the CICS Resource Definition Online (RDO) transaction, CEDA, are shown in Figure 11.

```
DEFINE GROUP(<group name 1>)
CONNECTION(<remote conn>)
NETNAME(<remote luname>)
ACCESSMETHOD(VTAM)
PROTOCOL(APPC)
SINGLESESS(NO)

DEFINE GROUP(<group name 1>)
SESSIONS(<sess name>)
CONN(<remote conn>)
MODE(<logmode 1>)
MAXIMUM(<max sessions>,<max CICS contention winners>)

INSTALL GROUP(<group name 1>)

ADD GROUP(<group name 1>) LIST(<start-up list>) {AFTER(<group name>)}
```

Figure 11. Definitions in CICS using RDO for parallel session partner LU

If the remote LU is capable of only one session, then it may be defined to CICS as either a single-session connection definition or as a terminal definition (Figure 13 on page 45).

```
DEFINE GROUP(<group name 2>)
CONNECTION(<remote conn>)
NETNAME(<remote luname>)
ACCESSMETHOD(VTAM)
PROTOCOL(APPC)
SINGLESESS(YES)

DEFINE GROUP(<group name 2>)
SESSIONS(<sess name>)
CONN(<remote conn>)
MODE(<logmode 2>)
MAXIMUM(1,1)

INSTALL GROUP(<group name 2>)

ADD GROUP(<group name 2>) LIST(<start-up list>) {AFTER(<group name>)}
```

Figure 12. Definitions in CICS for single-session capable partner LU

```
DEFINE GROUP(<group name 3>)
TERMINAL(<remote conn>)
NETNAME(<remote luname>)
TYPETERM(DFHLU62T)
MODENAME(<logmode 2>)

INSTALL GROUP(<group name 3>)

ADD GROUP(<group name 3>) LIST(<start-up list>) {AFTER(<group name>)}
```

Figure 13. Definitions in CICS singles-session capable LU

The CICS supplied typeterm definition, DFHLU62T, provides a suitable terminal type definition. It exists in group DFHTYPE, which should be installed on your system.

Sample definitions for CICS tables can be found in the sublibrary PRD2.WMQZVSE. However, other definitions are specific to your environment and you have to create them manually using the CEDA transaction, or DEFINE commands if using the DFHCSDUP batch program.

The definitions consist of a:
- Connection definition - see "Connection definition"
- Session definition - see "Session definition" on page 46

Connection definition

CICS uses the connection name to identify the other systems. For example, if sessions in VSE1 are to converse with sessions in VSE2 and z/OS, you must define both z/VSE and z/OS connections in each direction.

You must also define all the sessions and terminals involved if you are using SNA LU 6.2.

Type `CEDA DEF CONN GROUP(MQM)` to create connections, and set the fields to the following values:

Table 1. Object Characteristics of Connection

Category	Parameter	Desired Value
	Connection	VSE2
	Group	MQM
Connection Identifiers	Netname	vse2lu62
Connection Properties	ACcessmethod	Vtam
	Protocol	Appc
	Datastream	User
	RECordformat	U
Operational Properties	AUtoconnect	Yes
	INService	Yes
Security	ATtachsec	Local

The settings detailed, together with default values are sufficient for operation. For other parameters, refer to the *CICS for z/VSE 2.3 Resource Definition (Macro)* manual.

You can also display the connection status by typing `CEMT INQ CONN`, to display:

Table 2. CEMT I CONN display output.

STATUS: RESULTS - OVERTYPE TO MODIFY	
Conn(VSE2) Net(xxxxxxxx)	Ins Acq
Conn(z/OS) Net(xxxxxxxx)	Ins Rel

Session definition

Type `CEDA DEF SESSION G(MQM)` to create session names. Enter the values shown in Table 3 to complete the fields.

Table 3. CEDA V SESS display parameter settings

Category	**Parameter**	**Desired Value**
	Sessions	VSE1VSE2
	Group	MQM
Session Identifiers	Connection	VSE2
Session Properties	Protocol	Appc
	Maximum	00006,00003
	RECEIVEcount	No
	SENDCount	No
	SENDSize	04096
	RECEIVESize	04096
Operational Properties	Autoconnect	Yes
	Buildchain	Yes
	RELreq	No
	Discreq	No
Recovery	RECOvoption	Sysdefault

The settings detailed, together with default values, are sufficient for operation. For other parameters, refer to the *CICS for z/VSE 2.3 Resource Definition Guide*.

Note: The DFHSIT Table must have the parameter ISC = YES to make the WebSphere MQ system work.

WebSphere MQ for z/VSE configuration guidelines

The following guidelines refer to the WebSphere MQ master terminal (MQMT) administration dialogs. For information about using MQMT, see "WebSphere MQ master terminal (MQMT) – main menu" on page 79.

There are three levels of configuration immediately relevant to network communications:
- The queue manager.
- The channel.
- The queue.

Some fields are the same in all three levels, for example, the Maximum Message Size.

Note:

1. The maximum message size defined in the queue manager configuration must be the largest of all those defined in the channels for this queue manager.
2. The size defined in the channel configuration must be equal to, or greater than, the largest message size that is accessing this channel.
3. Each level of maximum message size configuration utilizes different kinds of resources. Unnecessarily large sizes will consume address space.

Queue manager configuration guidelines

When configuring the queue manager (see "Global system definition" on page 82), use the following guidelines:

Maximum Connection Handles
The maximum number (integer) of simultaneous connections to the queue manager. Though there is a slight overhead for each unused reservation, there is no harm in setting a large number, for example, 200.

Maximum Concurrent Queues
The maximum number of simultaneous open local queues allowed for the queue manager. You are recommended to set this to a large number, for example, 200.

System Wait Interval
The maximum polling time (in seconds) for the system monitor program after the system starts. A value of thirty seconds is usually sufficient.

Note: The system monitor task remains active until the queue manager or CICS region is shut down, but exists in a wait state until the task is activated by the expiration of the System Wait Interval or by some specific application interface tasks.

The system monitor task starts up the trigger program and schedules the processes that reclaim resources held by applications that have ended abnormally. If there are too many, the System Wait Interval should be reduced to schedule this cleanup process more frequently.

Maximum Q Depth
The maximum number of active messages allowed by the queue manager for each queue. This value serves as the default Maximum Q Depth value when defining a queue. Any inbound message that causes the queue depth to exceed this size will be rejected as "Queue Full".

If this value is smaller than the Maximum Q Depth specified in the queue definition, it becomes the limiting value for the queue. You should set the value to double the maximum number of messages expected to be queued before any application starts to process them.

Maximum Message Size
The maximum number of characters allowed by the queue manager for each message. This field needs only to be large enough to accommodate the largest message. Setting a higher value than necessary wastes resource.

For example, if you anticipate the largest message to be 10 KB (10,240 bytes) you should set this field to 10240.

Note: Messages are stored in VSAM clusters and large messages can span multiple VSAM records. However, you should avoid spanning multiple records wherever possible, because of performance implications.

Where an entire message is stored within a single VSAM record, a message header of 736 bytes, for identification and description, is prefixed to the message.

Where a message is split across multiple records, each subsequent record uses a 56-byte header as a prefix to the data.

Maximum Single Q Access
This field defines the maximum number of MQOPEN calls against any queue handled by this queue manager. A value of 100 calls is an acceptable value, if the maximum number of opens for each queue in the system is 100.

Maximum Global Locks
The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls, for each queue in the system, for recovery. A value of 500 is normally used.

Maximum Local Locks
The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls for each queue and task for recovery. Since an entry of a local lock is deleted once an application issues an explicit SYNCPOINT CICS command to commit updates, the more often an application takes the checkpoint, the fewer the maximum number of local locks needed. You should specify a value greater than the largest message batch size for all the channel records. A value of 200 is usually sufficient.

Channel configuration guidelines

Defining the remote WebSphere MQ system to the local queue manager is described in “Channel definitions” on page 112. However, from the point of view of showing where fields in the various definitions have to correspond, an outline WebSphere MQ channel definition is shown in Figure 14 on page 49.

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:10:50                    Channel Record           DISPLAY         CIC1
MQWMCHN                                                              A001
Channel  : MY.SDR
 Desc. . :
 Protocol: T (L/T)  Type : S (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000000
 Get retry number . . . . . : 00000000   Short retry interval  . . : 000000000
 Get retry delay (secs) . . : 00000000   Long retry interval . . . : 000000000
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000000000
 Property control . . . . . : C
 Transmission queue name. . : MY.XMITQ
 TP name. . :
Sender/Receiver/Server/Requester
 Connection : 1.1.1.1(1414)
 Max Messages per Batch . . : 000010     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 065535     Split Msg(Y/N)  . . . . . : Y
 Max TCP/IP Wait  . . . . . : 000300     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

Figure 14. Outline WebSphere MQ channel definition

When configuring the channel, use the following guidelines:

Protocol

The required transport options for this channel. The options are:

- L - LU 6.2 (SNA)
- T - TCP/IP

Remote TCP/IP port

The port number; relevant for TCP/IP defined channels only.

This field is relevant for sender, server and requester channels. Receiver channels are started by the WebSphere MQ listener program which uses the port number configured in the global system definition.

This field is ignored if the port number is specified as part of the connection name.

Type Channel type of (S)ender, (R)eceiver, ser(V)er, re(Q)uester, or svr(C)onn.

Connection

The channel partner name. This is the CICS connection ID for LU 6.2 channels, or the remote hostname or IP address for TCP/IP channels.

For TCP/IP this field is relevant for sender, server and requester channels. Sender channels identify a specific host for communications, whereas receiver channels can accept communications from any host.

Note that for TCP/IP channels, the connection name does not include a port number as it may do on other MQ platforms. The port number is configured as a separate channel parameter.

For TCP/IP sender channels, the remote port number can be appended (in parentheses) to the connection name. For example:

```
my.remote.host(1414)
```

If the port number is not appended to the connection name, the queue manager will use the value specified by the Remote TCP/IP port parameter.

Short/Long retry count
The retry count field represents the number of times an allocation is retried when the conversation has not been established. You should set the retry count at less than 10. If this value is exceeded, the system can be placed under stress.

For receiver channels, this value should be set to zero.

Short retry interval
The time interval, in seconds, that an allocation of conversation is retried for the first cycle of retries. A value of one to five seconds is sufficient for this field, with the longer time being used for a slow environment, for example, a dial-up SDLC.

For receiver channels, this value should be set to zero.

Long retry interval
The time interval, in seconds, that an allocation of conversation is retried for the next cycle of retries, should the first cycle of retries fail. A value between three and 10 seconds is sufficient for this field, with the longer time being used for a slow environment.

For receiver channels, this value should be set to zero.

Get retry number
The number of retries for the MQGET call when the queue is depleted. If a transmission queue is empty, the queue manager retries at the Delay-Time interval before disconnecting the channel or making a request to disconnect the channel.

For receiver channels, this value should be set to zero.

Get retry delay
The time interval, in seconds, between retries. The value of this field may depend on the size of message and the platforms where the LU resides. The optimum value can vary from 1 to 20 seconds. Sender channels process messages as they arrive, that is, the channel does not wait the full delay interval if messages have arrived and are ready for transmission.

The longer the delay time specified, the less frequently a channel is reopened. For time-consuming dial-up connections, you are recommended to use a value of 20 seconds.

For receiver channels, this value should be set to zero.

Note: By using a value of zero for the Number of Retries, and a value of "n" seconds for the Delay Time it is possible for you to set a simple disconnection interval similar to that provided on other WebSphere MQ platforms.

Max Messages per Batch
The maximum number of messages in the batch.

Message Sequence Wrap
The message sequence number (MSN) wrap count represents the highest MSN value used on this channel, after which the MSN reverts to one. You are recommended to set this value to 999 999 999.

Note: The value of the MSN Wrap count must be the same at both the sending and receiving ends of the channel.

Max Transmission Size
The mutually accepted maximum number of characters for each transmission. The minimum value should be equal to the maximum message size expected on this channel, plus 476 bytes for the transmission header.

The maximum transmission size for LU6.2 channels is 32000. For TCP/IP the maximum is 65535.

Max Message Size
The maximum number of bytes for each message that is allowed for this channel.

Convert Msgs
A field that identifies whether message data is converted before it is sent to a remote queue manager. To convert message data, set this field to Y.

Split Msg
A field that identifies whether message data can be split across network transmissions. For example, if the transmission size is 8 KB and message data lengths are up to 30 KB, then the message data must be split across transmissions.To split message data in such situations, set this field to Y.

TP Name
The remote task ID, character only, of the receiver on a remote CICS region or a Transaction Program name on a remote system. This is required by the sender, and since CICS uses four bytes as the transaction identifier, only the first four bytes of the remote task ID are meaningful for CICS to CICS conversation.

This field is not relevant for TCP/IP channels.

Note: z/VSE converts the name to uppercase, therefore, the corresponding name on the remote system should be defined in uppercase characters.

Max TCP/IP Wait
The maximum number of second that a Message Channel Agent (MCA) should wait to receive TCP/IP data before terminating the connection with an error. See “Bullet-proof channels” on page 74 for more information.

Channel statistics
Indicates whether or not channel statistics should be collected for the channel.

Channel monitoring
Indicates whether or not channel monitoring information should be collected for the channel.

Queue configuration guidelines

Defining queues to the local queue manager is described in “Queue definitions” on page 96. Certain parameters in queue definitions are important when configuring network communications. The queue extended definition, shown in Figure 15 on page 52, includes these parameters.

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:11:28               Queue Extended Definition                    CIC1
MQWMQUE                                                             A001

Object Name: MY.XMITQ

General               Maximums                    Events
Type   . . : Local    Max. Q depth . : 00010000  Service int. event: N
File name  : MQFI001  Max. msg length: 00040000  Service interval  : 00000000
Usage  . . : T        Max. Q users . : 00000100  Max. depth event  : N
Shareable  : Y        Max. gbl locks : 00000100  High depth event  : N
Dist.Lists : N        Max. lcl locks : 00000100  High depth limit  : 000
PropCtl. . : C                                   Low depth event . : N
Triggering                                       Low depth limit . : 000
Enabled  . : Y        Transaction id.:
Type . . . : E        Program id . . : MQPSEND
Max. starts: 0001     Terminal id  . :
Restart  . : N        Channel name . : MY.SDR
User data  :
           :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
            PF9=List  PF10=Queue
```

Figure 15. Outline WebSphere MQ extended queue definition

When configuring the queue (see "Queue definitions" on page 96), use the following guidelines:

File name
The CICS file name, of up to seven characters, used to store messages for this queue. It is recommended that files host only one WebSphere MQ queue. Once a queue has been defined, the file name cannot be changed. To move a queue to a different file, existing messages should be processed, the queue deleted, and redefined in with the new file name.

Note: You cannot use the MQFCNFG or the MQFREOR file for queue definitions. The MQFCNFG file is used by WebSphere MQ to store system configuration and cannot be used to host queue messages. The MQFREOR file is used by the automatic VSAM reorganization feature and is deleted and redefined during reorganization. WebSphere MQ checks this file before commencing a scheduled reorganization. If queue messages are present and the file defined to hold these messages does not exist, then the reorganization cannot take place, effectively disabling the reorganization feature.

Max. Q depth
The maximum number of records that can remain unread on this queue. Any inbound message that causes the queue depth to exceed this size is rejected as "Queue Full". The minimum value you set should be the maximum number of messages on the queue before the application starts to read and process the queue. In practice, you can set this to 9,999,999.

Max. msg length
The maximum number of characters for each message that this queue allows. If this queue is a transmission queue, the value needs to be sufficiently large to accommodate all messages using this queue as the outbound queue.

Max. Q users
The maximum number of MQOPEN calls that can occur on this queue. You are recommended to set a value of 100 for each queue that is not a transmission queue. For a transmission queue you should add a value of 100 calls, to the base of 100 calls, for each additional target queue that receives messages from this transmission queue. Setting a high value can use too much overhead.

Max. gbl locks
The maximum number of entries that the queue manager uses to maintain committed MQPUT and MQGET calls for this queue for system recovery. If the queue is intended for random message retrieval, rather than sequential processing, then specify a higher value (for example, 1000). For sequential processing, a lower value (for example, 200) should be sufficient.

Max. lcl locks
The maximum number of entries that the queue manager uses to maintain uncommitted MQPUT and MQGET calls for this queue for recovery. Since an entry of a local lock is deleted once an application issues an explicit SYNCPOINT CICS command to commit updates, the more often an application takes the checkpoint, the fewer the maximum number of local locks needed. A value similar to the Max. gbl locks setting is recommended.

Trigger Type
"F" is used to generate a trigger when an MQPUT call changes the status of a queue from empty to nonempty. The triggered transaction must have sufficient logic to empty the queue, including messages that may arrive during the process, in a single thread. "E" is used to generate a trigger whenever an MQPUT call occurs and may have as many threads as specified in Max Trigger Starts.

Max. starts
The maximum number of trigger threads that can be activated simultaneously. This field applies to Trigger Type "E" only.

Transaction id
The transaction to be started by the trigger. This field is mutually exclusive with the Program ID. You are recommended to leave this field blank and use a Program ID, for example MQPSEND, unless you require a user transaction.

Once the initial maximum trigger starts is reached then WebSphere MQ for z/VSE only checks that the maximum trigger starts are running at every system interval and not when each task completes. If it is important to have a definite number of trigger instances running against a queue, you should use Program ID to identify your trigger program.

Program id
You should use the MQPSEND call on a transmission queue if you require triggering.

Terminal id
You should leave this field blank unless you require a terminal for problem determination purposes.

Channel Name
This field should be left blank except for a transmission queue definition. For a transmission queue definition, this field must identify a channel name.

User Data
This field is for static data that you want to pass to the trigger instance. When a trigger instance is activated, it is passed data in the form of the MQTM structure (see the CMQTML and CMQTMV copybooks). Data in the User Data field is passed in the MQTM-USERDATA field.

Event The event settings for queues do not affect network communications. For a full description of the event settings, refer to "Local queue extended definition screen" on page 100.

Permitted number of channels

The limit on the number of channels depends upon the availability of system resources. The queue manager can support as many channels and transmission queues as the resource in the system permits, up to a maximum of 1000 channels and 1000 queues.

Example configuration

The following tables give a set of values that can be used to set up your system. See:
- Table 4 for the queue manager.
- Table 5 for a channel.
- Table 6 on page 55 for a queue.

Table 4. Example queue manager configuration

Parameter	Value	Units
Maximum Number of MQCONN	200	integer
Maximum Open Queue	200	integer
System Wait Interval	30	seconds
Maximum Q Depth	9999999	integer
Maximum Message Size	5000	bytes
Maximum Number of Opens	500	integer
Max Number of Global Locks	500	integer
Max Number of Local Locks	500	integer

Table 5. Example channel configuration

Parameter	Value	Units
Short/Long retry count	4	integer
Short retry interval	1	second
Long retry interval	3	seconds
Get Retries	1	integer
Delay Time	10	seconds
Message Sequence Wrap	999999999	integer
Maximum Transmission Size	3821	bytes
Maximum Message Size	5000	bytes

Table 6. Example queue configuration

Parameter	Value	Units
Maximum Q Depth	999999	integer
Maximum Message Size	5000	bytes
Maximum Number of Opens	100	integer
Max Number of Global Locks	100	integer
Max Number of Local Locks	100	integer
Trigger Type	E	character
Maximum Trigger Starts	1	integer
Transaction Id	<blank>	character
Program Id	MQPSEND (transmit queue) user app. (other queues)	character

Channel exits

Channel-exit programs are called at defined places in the processing carried out by WebSphere MQ Message Channel Agent (MCA) programs.

Some of these user-exit programs work in complementary pairs. For example, if a user-exit program is called by the sending MCA to encrypt the messages for transmission, the complementary process must be functioning at the receiving end to reverse the process.

The different types of channel-exit program include:
Security exit.
Send exit.
Receive exit.
Message exit.
Message retry exit.
Auto-definition exit.
Transport retry exit.

WebSphere MQ for z/VSE does not support message retry or transport retry exits since these exits involve features of MQ that are not supported by WebSphere MQ for z/VSE.

Channel security exits

You can use security exit programs to verify that the partner at the other end of a channel is genuine.

Channel security exit programs are called at the following places in an MCA's processing cycle:

- At MCA initiation and termination.
- Immediately after the initial data negotiation is finished on channel startup. The receiver end of the channel may initiate a security message exchange with the remote end by providing a message to be delivered to the security exit at the remote end. It may also decline to do so. The exit program is re-invoked to process any security message received from the remote end.
- Immediately after the initial data negotiation is finished on channel startup. The sender end of the channel processes a security message received from the remote

end, or initiates a security exchange when the remote end cannot. The exit program is re-invoked to process all subsequent security messages that may be received.

Channel send and receive exits

You can use the send and receive exits to perform tasks such as data compression and decompression. In WebSphere MQ for z/VSE you can configure a chain of up to 8 send and 8 receive exits.

Channel send and receive exit programs are called at the following places in an MCA's processing cycle:

- The send and receive exit programs are called for initialization at MCA initiation and for termination at MCA termination.
- The send exit program is invoked at either end of the channel, immediately before a transmission is sent over the link.
- The receive exit program is invoked at either end of the channel, immediately after a transmission has been taken from the link.

There may be many transmissions for one message transfer, and there could be many iterations of the send and receive exit programs before a message reaches the message exit at the receiving end.

The channel send and receive exit programs are passed an agent buffer containing the transmission data as sent or received from the communications link. For send exit programs, the first eight bytes of the buffer are reserved for use by the MCA, and must not be changed. If the program returns a different buffer, then these first eight bytes must exist in the new buffer. The format of data presented to the exit programs is not defined.

A good response code must be returned by send and receive exit programs. Any other response will cause an MCA abnormal end (abend). Since data traffic can continue after an abnormal end (so as to communicate the channel failure to the remote MCA), send and receive exits are automatically suppressed following an bad response code. However, the exit is still called at termination of the channel.

Send and receive exits usually work in pairs. For example a send exit may compress the data and a receive exit decompress it, or a send exit may encrypt the data and a receive exit decrypt it. When you define the appropriate channels, make sure that compatible exit programs are named for both ends of the channel.

Channel send and receive exits may be called for message segments other than for application data, for example, status messages. They are not called during the startup dialog, nor the security check phase.

Although message channels send messages in one direction only, channel-control data flows in both directions, and these exits are available in both directions, also. However, some of the initial channel startup data flows are exempt from processing by any of the exits.

There are circumstances in which send and receive exits could be invoked out of sequence; for example, if you are running a series of exit programs or if you are also running security exits. Then, when the receive exit is first called upon to process data, it may receive data that has not passed through the corresponding

send exit. If the receive exit were just to perform the operation, for example decompression, without first checking that it was really required, the results would be unexpected.

You should code your send and receive exits in such a way that the receive exit can check that the data it is receiving has been processed by the corresponding send exit. The recommended way to do this is to code your exit programs so that:

- The send exit sets the value of the ninth byte of data to 0 and shifts all the data along one byte, before performing the operation. (The first eight bytes are reserved for use by the MCA.)
- If the receive exit receives data that has a 0 in byte 9, it knows that the data has come from the send exit. It removes the 0, performs the complementary operation, and shifts the resulting data back by one byte.
- If the receive exit receives data that has something other than 0 in byte 9, it assumes that the send exit has not run, and sends the data back to the caller unchanged.

When using security exits, if the channel is ended by the security exit it is possible that a send exit may be called without the corresponding receive exit. One way to prevent this from being a problem is to code the security exit to set a flag, in MQCD.SecurityUserData or MQCD.SendUserData, for example, when the exit decides to end the channel. Then the send exit should check this field, and process the data only if the flag is not set. This prevents the send exit from unnecessarily altering the data, and thus prevents any conversion errors that could occur if the security exit received altered data.

In the case of MQI channels for clients, byte 10 of the agent buffer identifies the API call in use when the send or receive exit is called. This is useful for identifying which channel flows include user data and may require processing such as encryption or digital signing.

The following table shows the data that appears in byte 10 of the channel flow when an API call is being processed (note that these are not the only values of this byte; there are other reserved values):

Table 7. Identifying API calls

API call	Value of byte 10
MQCONN request[1, 2]	X'81'
MQCONN reply[1, 2]	X'91'
MQDISC request[1]	X'82'
MQDISC reply[1]	X'92'
MQOPEN request	X'83'
MQOPEN reply	X'93'
MQCLOSE request	X'84'
MQCLOSE reply	X'94'
MQGET request[3]	X'85'
MQGET reply[3]	X'95'
MQPUT request[3]	X'86'
MQPUT reply[3]	X'96'
MQPUT1 request[3]	X'87'

Table 7. Identifying API calls (continued)

MQPUT1 reply[3]	X'97'
MQSET request	X'88'
MQSET reply	X'98'
MQINQ request	X'89'
MQINQ reply	X'99'
MQCMIT request	X'8A'
MQCMIT reply	X'9A'
MQBACK request	X'8B'
MQBACK reply	X'9B'

Note:

1. The connection between the client and server is initiated by the client application using MQCONN. Therefore, for this command in particular, there will be several other network flows. This also applies to MQDISC that terminates the network connection.
2. MQCONNX is treated in the same way as MQCONN for the purposes of the client-server connection.
3. If the message data exceeds the transmission segment size, there may be a large number of network flows per single API call.

Channel message exits

You can use the channel message exit for the following:

- Encryption on the link.
- Validation of incoming user IDs.
- Substitution of user IDs according to local policy.
- Message data conversion.
- Journaling.
- Reference message handling.

In WMQ for z/VSE you can configure a chain of up to 8 message exits. Message exits are ignored for server-connection channels.

Channel message exit programs are called at the following places in an MCA's processing cycle:

- At MCA initiation and termination.
- Immediately after a sending MCA has issued an MQGET call.
- Before a receiving MCA issues an MQPUT call.

The message exit is passed an agent buffer containing the transmission queue header, MQXQH, and the application message text as retrieved from the queue. (The format of MQXQH is given in "MQXQH – Transmission-queue header" on page 866.) If you use reference messages, that is messages that contain only a header which points to some other object that is to be sent, the message exit recognizes the header, MQRMH. It identifies the object, retrieves it in whatever way is appropriate appends it to the header, and passes it to the MCA for transmission to the receiving MCA. At the receiving MCA, another message exit recognizes that this is a reference message, extracts the object, and passes the header on to the destination queue. See the WebSphere MQ Application Programming Guide for more information about reference messages and some sample message exits that handle them.

Message exits can return the following responses:

- Send the message (GET exit). The message may have been changed by the exit. (This returns MQXCC_OK.)
- Put the message on the queue (PUT exit). The message may have been changed by the exit. (This returns MQXCC_OK.)
- Do not process the message. The message is placed on the dead-letter queue (undelivered message queue) by the MCA.
- Close the channel.
- Bad return code, which causes the MCA to abend.

Message exits are called just once for every complete message transferred, even when the message is split into parts. An exit runs in the same thread as the MCA itself. It also runs inside the same unit of work (UOW) as the MCA because it uses the same connection handle. Therefore, any calls made under syncpoint are committed or backed out by the channel at the end of the batch. For example, one channel message exit program can send notification messages to another and these messages will only be committed to the queue when the batch containing the original message is committed.

Therefore, it is possible to issue syncpoint MQI calls from a channel message exit program.

Channel auto-definition exit

In WebSphere MQ for z/VSE, when a remote system attempts to connect to the z/VSE queue manager as a sender or client, if there is no appropriate channel definition, then a definition is created automatically if the queue manager's channel auto-definition attribute is enabled.

The definition is created using:

1. The appropriate model channel definition, SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCONN. The model channel definitions for auto-definition are the same as the system defaults, SYSTEM.DEF.RECEIVER and SYSTEM.DEF.SVRCONN, except for the description field, which is "Auto-defined by" followed by 49 blanks. The systems administrator can choose to change any part of the supplied model channel definitions.
2. Information from the partner system. The partner's values are used for the channel name and the sequence number wrap value.
3. A channel exit program, which you can use to alter the values created by the auto-definition, or stop the channel from being defined.

The description is then checked to determine whether it has been altered by an auto-definition exit or because the model definition has been changed. If the first 44 characters are still "Auto-defined by" followed by 29 blanks, the queue manager name is added.

Once the definition has been created and stored the channel start proceeds as though the definition had always existed. The batch size, transmission size, and message size are negotiated with the partner.

The auto-defined channel persists after the channel is closed.

Configuring channel exits

Channel exits, and their associated exit data, can be configured using:

- Master Terminal transaction (MQMT).
- Programmable Command Formats (PCF).
- WebSphere MQ Commands (MQSC).
- WebSphere MQ Explorer.

Configuration using MQMT

Channel definitions can be created and modified using the master master terminal transaction, MQMT option 1.3, “Channel Definitions”.

The Channel Definitions screen appears as follows:

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:12:07                    Channel Record           DISPLAY         CIC1
MQWMCHN                                                              A001
Channel  : MY.SDR
 Desc. . :
 Protocol: T (L/T)  Type : S (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000000
 Get retry number . . . . . : 00000000   Short retry interval  . . : 000000000
 Get retry delay (secs) . . : 00000000   Long retry interval . . . : 000000000
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000000000
 Property control . . . . . : C
 Transmission queue name. . : MY.XMITQ
 TP name. . :
Sender/Receiver/Server/Requester
 Connection : 1.1.1.1(1414)
 Max Messages per Batch . . : 000010     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 065535     Split Msg(Y/N)  . . . . . : Y
 Max TCP/IP Wait  . . . . . : 000300     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

Figure 16. Channel Definitions screen

From this screen, PF11 activates the Channel Exit Settings screen, which appears as follows:

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:54:10                  Channel Exit Settings                     CIC1
MQWMCHN                                                             A000

Channel name . . . : VSE9.IP.ZOSX
Channel type . . . : Sender

Security exit name : MQPCHNX
Security exit data :









Channel exit settings displayed.
F2=Return  PF3=Quit  PF4=Read  F6=Update    PF10=SndX PF11=RcvX PF12=MsgX
```

Figure 17. Channel Exit Settings screen

The Channel Exit Settings screen allows channel exit programs, and associated data, to be set for send, receive, security and message exits. Send, Receive and Message exits are configurable using function keys PF10, PF11 and PF12 respectively.

Channel exit names can be 1-8 characters, and follow the naming standard for any program defined in the CICS CSD.

Channel exit data can be 1-32 characters and is optional. Data specified in the channel definition is passed to exit programs when they are invoked in the Channel Definition (MQCD) data structure.

Configuration for send, receive and message exits is similar. For example, using PF10 to configure send exits presents the following screen:

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:59:33               Channel Send Exit Settings                   CIC1
MQWMCHN                                                             A000

Channel name . . . : SYSTEM.DEF.SENDER
Channel type . . . : Sender

  Exitname           Exit Data
> SNDEXIT1 <       > Send exit data 1                <
> SNDEXIT2 <       > Send exit data 2                <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <




Channel exit settings displayed.
F2=Return  PF3=Quit  PF4=Read  F6=Update    PF10=SndX PF11=SecX PF12=MsgX
```

Figure 18. Channel Send Exit Settings screen

Note that you can configure up to 8 send exits and associate exit data. These exits, if configured, are called in the sequence listed.

Channel exit names can be 1-8 characters, and follow the naming standard for any program defined in the CICS CSD.

Configuring auto-definition with MQMT

The channel auto-definition exit is a queue manager attributes and is specified in the Global System Definition, accessible via MQMT option 1.1, and PF9.

```
11/17/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
13:14:17                  Global System Definition                  CIC1
MQWMSYS                    Communications Settings                  A000

    TCP/IP settings                          Batch Interface settings
    TCP/IP listener port : 01460             Batch Int. identifier: MQBISRVA
    Licensed clients . . : 00000             Batch Int. auto-start: Y
    Adopt MCA  . . . . . : N
    Adopt MCA Check  . . : N                 Channel Auto-Definition
                                             Auto-definition  . . : N
    SSL parameters                           Auto-definition exit :
    Key-ring sublibrary  : MQM.SSLKEYS
    Key-ring member  . . : MQVSED
    SSL reset count  . . : 001024000

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : N

Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 19. Channel auto-definition exit screen

The channel auto-definition exit is only called if the Auto-definition attribute is set to (Y)es. If so, the channel auto-definition exit program is called when a connection request is received for a channel that does not already exist. At this point, the channel can change the definition details, or stop the auto-definition from continuing.

Channel exit names can be 1-8 characters, and follow the naming standard for any program defined in the CICS CSD.

Configuration using PCF

Channel definitions can be created and modified using the Programmable Command Formats (PCF).

Channel exits, and their associated data, can be manipulated using the following PCF commands:

- Create Channel
- Change Channel
- Copy Channel
- Inquire Channel

For each of these commands, the following parameters are supported:

SecurityExit (MQCFST)
: Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME). The maximum length of the exit name is restricted to the MQ_EXIT_NAME_LENGTH constant.

MsgExit (MQCFST)
: Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME). The maximum length of the exit name is restricted to the MQ_EXIT_NAME_LENGTH constant.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME). The maximum length of the exit name is restricted to the MQ_EXIT_NAME_LENGTH constant.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME). The maximum length of the exit name is restricted to the MQ_EXIT_NAME_LENGTH constant.

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA). The maximum length of the exit user data is restricted to the MQ_EXIT_DATA_LENGTH constant.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA). The maximum length of the exit user data is restricted to the MQ_EXIT_DATA_LENGTH constant.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA). The maximum length of the exit user data is restricted to the MQ_EXIT_DATA_LENGTH constant.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA). The maximum length of the exit user data is restricted to the MQ_EXIT_DATA_LENGTH constant.

If you are configuring chained exits, that is more than one send, receive or message exit, you must use the MQCFSL structure rather than the MQCFST structure to list the exit values for both the exit name and the exit data.

Configuring auto-definition with PCF

The channel auto-definition exit is a queue manager attributes and can be manipulated using the following PCF commands:

- Change Queue Manager.
- Inquire Queue Manager.

For each of these commands, the following parameters are supported:

ChannelAutoDef (MQCFIN)
Controls whether automatic channel definition is permitted. This attribute controls the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. The value is one of the following:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

Note that channel auto-definition cannot occur even if this attribute is enabled if the model channel SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCONN does not exist.

ChannelAutoDefExit (MQCFST)
Name of user exit for automatic channel definition.

If this name is nonblank, and ChannelAutoDef has the value MQCHAD_ENABLED, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. The exit can then do one of the following:

- Allow the creation of the channel definition to proceed without change.
- Modify the attributes of the channel definition that is created.
- Suppress creation of the channel entirely.

Both the length and the value of this attribute are environment specific. For WebSphere MQ for z/VSE, an exit name can be up to eight characters.

Configuration using MQSC

Channel definitions can be created and modified using WebSphere MQ Commands (MQSC).

Channel exits, and their associated data, can be manipulated using the following MQSC commands:

- DEFINE CHANNEL
- ALTER CHANNEL
- DISPLAY CHANNEL

For each of these commands, the following parameters are supported:

SCYEXIT (string)
Channel security exit name. For WebSphere MQ for z/VSE, exit names are 1-8 characters.

MSGEXIT (string)
Channel message exit name. For WebSphere MQ for z/VSE, exit names are 1-8 characters.

SENDEXIT (string)
Channel send exit name. For WebSphere MQ for z/VSE, exit names are 1-8 characters.

RCVEXIT (string)
Channel receive exit name. For WebSphere MQ for z/VSE, exit names are 1-8 characters.

SCYDATA (string)
Channel security exit user data. For WebSphere MQ for z/VSE, exit user data can be 0-32 characters.

MSGDATA (string)
Channel message exit user data. For WebSphere MQ for z/VSE, exit user data can be 0-32 characters.

SENDDATA (string)
Channel send exit user data. For WebSphere MQ for z/VSE, exit user data can be 0-32 characters.

RCVDATA (string)
Channel receive exit user data. For WebSphere MQ for z/VSE, exit user data can be 0-32 characters.

If you are configuring chained exits, that is more than one send, receive or message exit, you can specify a list of values, for example:

```
SENDEXIT(name, name, name,...)
SENDDATA(data, data, data,...)
```

Configuring auto-definition with MQSC

The channel auto-definition exit is a queue manager attribute and can be manipulated using the following MQSC commands:

- ALTER QMGR
- DISPLAY QMGR

For each of these commands, the following parameters are supported:

CHAD (ENABLE/DISABLE)
Whether receiver and server-connection channels can be defined automatically:

DISABLED
Auto-definition is not used. This is the queue manager's initial default value.

ENABLED
Auto-definition is used.

CHADEXIT(string)
Auto-definition exit name.

If this name is nonblank, the exit is called when an inbound request for an undefined receiver or server-connection channel is received.

The format and maximum length of the name depends on the environment. On z/VSE, the exit name is eight characters and names a program defined to CICS.

Configuration using WebSphere MQ Explorer

As an alternative to the master terminal transactions, PCF and MQSC, the z/VSE queue manager can be configured using the WebSphere MQ Explorer.

The WMQ Explorer, available on Windows and Linux (x86) platforms, uses PCF internally. Consequently, you must configure and start the PCF command server before attempting to administer WMQ for z/VSE using the Explorer.

For more information about the PCF and the command server, refer to “Preparing WebSphere MQ for PCF” on page 235. For more information about the WebSphere MQ Explorer, refer to “Administration using the WebSphere MQ Explorer” on page 154.

Writing and compiling channel-exit programs

Channel exits must be named in the channel definition. You can do this when you first define the channels, or you can add the information later using, for example, the MQSC command ALTER CHANNEL. The format of the exit name must comply with the naming standards for program entries defined in the CICS CSD.

If the channel definition does not contain a user-exit program name, a user exit is not called.

User exits and channel-exit programs are able to make use of all MQI calls, except as noted in the sections that follow. To get the connection handle, an MQCONN must be issued, even though a warning, MQRC_ALREADY_CONNECTED, is returned because the channel itself is connected to the queue manager.

Note: You are recommended to avoid issuing the following MQI calls in channel-exit programs:

- MQCMIT
- MQBACK

An exit runs in the same thread as the MCA itself and uses the same connection handle. So, it runs inside the same UOW as the MCA and any calls made under syncpoint are committed or backed out by the channel at the end of the batch.

Therefore, a channel message exit could send notification messages that will only be committed to that queue when the batch containing the original message is committed. So, it is possible to issue syncpoint MQI calls from a channel message exit.

Channel-exit programs should not modify the Channel data structure (MQCD), except in the case that it is necessary to communicate with Other exit programs via associated user exit data.

Also, for programs written in C, non-reentrant C library function should not be used in a channel-exit program.

All exits are called with a channel exit parameter structure (MQCXP), a channel definition structure (MQCD), a prepared data buffer, data length parameter, and buffer length parameter. The buffer length must not be exceeded:

- For message exits, you should allow for the largest message required to be sent across the channel, plus the length of the MQXQH structure.
- For send and receive exits, the largest buffer you should allow for is 64 KB. Note: Receive exits on sender channels and sender exits on receiver channels use 2 KB buffers for TCP.
- For security exits, the distributed queuing facility allocates a buffer of 1000 bytes.

It is permissible for the exit to return an alternate buffer, together with the relevant parameters. See “MQ_CHANNEL_EXIT - Channel exit” on page 68.

Exit programs in CICS

An exit program must be written in Language Environment (LE) C, COBOL, or PL/I. In CICS, the exits are invoked with EXEC CICS LINK with the Parameters passed by pointers (addresses) in the CICS communication Area (COMMAREA). The exit programs, named in the channel definitions, reside in a library in the LIBDEF SEARCH concatenation of the CICS startup JCL. They must be defined in the CICS system definition file CSD, and must be enabled.

User-exit programs can also make use of CICS API calls, but you should not issue syncpoints because the results could influence units of work declared by the MCA.

Any non-WebSphere MQ for z/VSE resources updated by an exit are committed, or backed out, at the next syncpoint issued by the channel program.

Channel-exit calls and data structures

This topic provides reference information about the special WebSphere MQ calls and data structures used when writing channel exit programs. This is product-sensitive programming interface information. You can write WebSphere MQ user exits in LE C, COBOL or PL/I.

In a number of cases, parameters are arrays or character strings whose size is not fixed. For these, a lowercase "n" is used to represent a numeric constant. When the declaration for that parameter is coded, the "n" must be replaced by the numeric value required. For further information about the conventions used in these descriptions, see the WebSphere MQ Application Programming Reference book.

The calls are:

MQ_CHANNEL_EXIT
Channel exit

The data structures are:

MQCD
Channel data structure

MQCXP
Channel exit parameter structure

MQ_CHANNEL_EXIT - Channel exit

This call definition is provided solely to describe the parameters that are passed to each of the channel exits called by the Message Channel Agent. No entry point called MQ_CHANNEL_EXIT is actually provided by the queue manager; the name MQ_CHANNEL_EXIT is of no special significance since the names of the channel exits are provided in the channel definition MQCD.

WebSphere MQ for z/VSE supports five types of channel exit:
- Channel security exit.
- Channel message exit.
- Channel send exit.
- Channel receive exit.
- Auto-definition exit.

The parameters are similar for each type of exit, and the description given here applies to all of them, except where specifically noted.

Syntax:

```
MQ_CHANNEL_EXIT (ChannelExitParms, ChannelDefinition, DataLength,
                 AgentBufferLength, AgentBuffer, ExitBufferLength,
                 ExitBufferAddr)
```

Parameters: The MQ_CHANNEL_EXIT call has the following parameters.

ChannelExitParms (MQCXP) - input/output
Channel exit parameter block. This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA should proceed.

ChannelDefinition (MQCD) - input/output
Channel definition. This structure contains parameters set by the administrator to control the behavior of the channel.

DataLength (MQLONG) - input/output
Length of data. When the exit is invoked, this contains the length of data in the AgentBuffer parameter. The exit must set this to the length of the data in either the AgentBuffer or the ExitBufferAddr (as determined by the ExitResponse2 field in the ChannelExitParms parameter) that is to proceed.

The data depends on the type of exit:

- For a channel security exit, when the exit is invoked this contains the length of any security message in the AgentBuffer field, if ExitReason is MQXR_SEC_MSG. It is zero if there is no message. The exit must set this field to the length of any security message to be sent to its partner if it sets ExitResponse to MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG. The message data is in either AgentBuffer or ExitBufferAddr.

  The content of security messages is the sole responsibility of the security exits.
- For a channel message exit, when the exit is invoked this contains the length of the message (including the transmission queue header). The exit must set this field to the length of the message in either AgentBuffer or ExitBufferAddr that is to proceed.
- For a channel send or channel receive exit, when the exit is invoked this contains the length of the transmission. The exit must set this field to the length of the transmission in either AgentBuffer or ExitBufferAddr that is to proceed.

  If a security exit sends a message, and there is no security exit at the other end of the channel, or the other end sets an ExitResponse of MQXCC_OK, the initiating exit is re-invoked with MQXR_SEC_MSG and a null response (DataLength=0).

AgentBufferLength (MQLONG) - input
Length of agent buffer. This can be greater than DataLength on invocation.

For channel message, send, and receive exits, any unused space on invocation can be used by the exit to expand the data in place. If this is done, the DataLength parameter must be set appropriately by the exit.

AgentBuffer (MQBYTE | AgentBufferLength) - input/output
Agent buffer. The contents of this depend upon the exit type:

For a channel security exit, on invocation of the exit it contains a security message if ExitReason is MQXR_SEC_MSG. If the exit wishes to send a security message back, it can either use this buffer or its own buffer (ExitBufferAddr).

For a channel message exit, on invocation of the exit this contains the transmission queue header (MQXQH), which includes the message descriptor (which itself contains the context information for the message), immediately followed by the message data.

If the message is to proceed, the exit can do one of the following:

- Leave the contents of the buffer untouched.
- Modify the contents in place (returning the new length of the data in DataLength; this must not be greater than AgentBufferLength).
- Copy the contents to the ExitBufferAddr, making any required changes

Any changes that the exit makes to the transmission queue header are not checked; however, erroneous modifications may mean that the message cannot be put at the destination.

For a channel send or receive exit, on invocation of the exit this contains the transmission data. The exit can do one of the following:

- Leave the contents of the buffer untouched.
- Modify the contents in place (returning the new length of the data in DataLength; this must not be greater then AgentBufferLength).

- Copy the contents to the ExitBufferAddr, making any required changes.

Note that the first 8 bytes of the data must not be changed by the exit.

ExitBufferLength (MQLONG) - input/output
Length of exit buffer. On the first invocation of the exit, this is set to zero. Thereafter whatever value is passed back by the exit, on each invocation, is presented to the exit next time it is invoked.

ExitBufferAddr (MQPTR) - input/output
Address of exit buffer. This is a pointer to the address of a buffer of storage managed by the exit, where it can choose to return message or transmission data (depending upon the type of exit) to the agent if the agent's buffer is or may not be large enough, or if it is more convenient for the exit to do so.

On the first invocation of the exit, the address passed to the exit is null. Thereafter whatever address is passed back by the exit, on each invocation, is presented to the exit the next time it is invoked.

Note that the auto-definition exit only receives the ChannelExitParms and ChannelDefinition parameters.

Usage notes: The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.

The ChannelDefinition parameter passed to the channel exit, for WebSphere MQ for z/VSE, is always MQCD_VERSION_7.

In general, channel exits are allowed to change the length of message data. This may arise as a result of the exit adding data to the message, or removing data from the message, or compressing or encrypting the message. However, special restrictions apply if the message is a segment that contains only part of a logical message. In particular, there must be no net change in the length of the message as a result of the actions of complementary sending and receiving exits.

For example, it is permissible for a sending exit to shorten the message by compressing it, but the complementary receiving exit must restore the original length of the message by decompressing it, so that there is no net change in the length of the message.

This restriction arises because changing the length of a segment would cause the offsets of later segments in the message to be incorrect, and this would inhibit the queue manager's ability to recognize that the segments formed a complete logical message.

CICS invocation: The WebSphere MQ MCA uses the CICS command level LINK call to pass control to the exit program.

The LINK call passes a communication area (COMMAREA) to the exit program that contains the addresses of the exit parameters as follows:

```
struct tagEXITPARMS
{
    MQCXP    *ChannelExitParms;
    MQCD     *ChannelDefinition;
    MQLONG   *DataLength;
    MQLONG   *AgentBufferLength;
```

```
    VOID     *AgentBuffer;
    MQLONG   *ExitBufferLength;
    VOID     *ExitBufferAddr;
} EXITPARMS;
```

MQCD - Channel data structure

The MQCD structure contains the parameters which control execution of a channel. It is passed to each channel exit that is called from a Message Channel Agent (MCA). See "MQ_CHANNEL_EXIT - Channel exit" on page 68.

The MQCD data structure is described in the WebSphere MQ Intercommunication manual.

MQCXP - Channel exit parameter structure

The MQCXP structure is passed to each type of exit called by a Message Channel Agent (MCA). See "MQ_CHANNEL_EXIT - Channel exit" on page 68.

The exit should not expect that any input fields that it changes in the channel exit parameter block will be preserved for its next invocation. Changes made to input/output fields (for example, the ExitUserArea field), are preserved for invocations of that instance of the exit only. Such changes cannot be used to pass data between different exits defined on the same channel, or between the same exit defined on different channels.

The MQCXP data structure is described in the WebSphere MQ Intercommunication manual.

Auto-definition exit and data structures

The channel auto-definition exit can be called when a request is received to start a receiver or server-connection channel, but no channel definition exists.

You can use it to modify the supplied default definition for an automatically defined receiver or server-connection channel, SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCONN.

Similar to other channel exits, the parameter list is:

- MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)

The ChannelDefinition parameter contains the values that are used in the default channel definition it they are not altered by the exit.

The channel auto-definition exit returns a response of either MQXCC_OK, or MQXCC_SUPPRESS_FUNCTION. If neither of these is returned, the MCA continues processing as thought MQXCC_SUPPRESS_FUNCTION were returned. That is, the auto-definition is abandoned, no new channel definition is created, and the channel cannot start.

MQCXP - Channel exit parameter structure

The MQCXP structure is passed to each type of exit called by a Message Channel Agent (MCA). See "MQ_CHANNEL_EXIT - Channel exit" on page 68.

When an auto-definition exit is called, the MQCXP field, ExitId, contains the following value:

MQXT_CHANNEL_AUTO_DEF_EXIT
Channel auto-definition exit.

The ExitREason field contains one of the following values:

MQXR_AUTO_RECEIVER
Auto-define receiver channel

MQXR_AUTO_SVRCONN
Auto-define server-connection channel

The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP data structure.

Channel exit sample

WebSphere MQ for z/VSE provides a sample channel exit with the installation library. The sample exit is a CICS COBOL program provided in sublibrary member MQPCHNX.Z.

The MQPCHNX sample can be used as a base for your own exit programs. It is generic in the sense that it includes logic to function as a security, send, receive and message exit. It achieves this by examining the contents of the ExitId field in the MQCXP data structure. The ExitId Field indicates which type of exit is being called. Depending on the exit type, the sample branches to appropriate logic.

Alternatively, the sample can be used as a base for individual exit programs that handle only one type of exit call, for example, the message exit. To use the sample in this way, additional logic that examines the ExitId can be removed.

Adopt MCA

The Adopt MCA feature is an integral feature of WebSphere MQ channel operation. It exists to solve a problem with Message Channel Agent (MCA) Receiver tasks falling into an indefinite wait state following a transport error.

When such an error occurs the receiver channel is often unaware of this and remains RUNNING even though the sender is RETRYING.

Once communication is re-established the retrying sender attempts to start a new receiver instance, but since a prior instance of this receiver still exists (because it didn't detect the communication failure), WebSphere MQ "believes" that there has been an invalid attempt to start multiple instances of the same receiver, from the same location, and accordingly treats this as an error, and fails the request.

This problem continues until either the original receiver instance detects the failure, or the channel is forcibly stopped.

Typically an WebSphere MQ receiver is waiting for messages from its sending partner. In the event of a network failure we would hope the receiver (which is effectively in a communication receive call) would be alerted to this by the communication subsystem. In some cases this is not possible and the receiver will continue running indefinitely, even though its partner MCA has ended.

This causes problems when the remote side attempts to re-establish the channel as WebSphere MQ finds the receiver is already running and prevents a duplicate instance from starting up. The channel cannot be restarted until either the operator has manually stopped the orphaned receiver or some communication timeout such as the TCP/IP keepalive timer causes the receiver to eventually fail.

The Adopt MCA feature allows an administrator to specify that WebSphere MQ should automatically stop an orphaned instance of a channel where it receives a new inbound connection request for that channel.

The administrator can specify the level of checking performed before an orphaned candidate is adopted based on combinations of the channel name (must always match for adoption), and the machine address. This allows for less rigorous checking in, for example a DHCP TCP environment where the partner machine's address may change frequently. Note that the Adopt MCA feature is applicable to TCP/IP channels only.

Review section "Features" on page 15 for prerequisites for this feature.

Adopt MCA parameters

Adoption can be enabled or disabled, and the level of checking can be set, via the queue manager's global system definition, MQMT option 1.1, PF9:

```
11/17/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
13:14:17                  Global System Definition                  CIC1
MQWMSYS                    Communications Settings                  A000

    TCP/IP settings                          Batch Interface settings
    TCP/IP listener port : 01460             Batch Int. identifier: MQBISRVA
    Licensed clients . . : 00000             Batch Int. auto-start: Y
    Adopt MCA  . . . . . : N
    Adopt MCA Check  . . : N                 Channel Auto-Definition
                                             Auto-definition  . . : Y
    SSL parameters                           Auto-definition exit : MQ9CHADX
    Key-ring sublibrary  : MQM.SSLKEYS
    Key-ring member  . . : MQVSED
    SSL reset count  . . : 001024000

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : N

Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 20. Communication Setting, Adopt MCA parameters

The communication setting parameters that affect Adopt MCA feature operation include:

Adopt MCA
Adopt MCA Check

Adopt MCA

The Adopt MCA parameter specifies whether or not an orphaned instance of a channel will be automatically restarted. Valid values include:

YES Automatically stop an orphaned MCA instance, if the appropriate Adopt MCA checks are met.

NO Do not automatically stop an orphaned MCA instance.

The default value is NO.

Activating the Adopt MCA feature by setting this parameter to (Y)es, applies to all TCP/IP Receiver channels.

Adopt MCA Check

The Adopt MCA Check parameter specifies whether the network address of the new MCA must be from the same address as the instance already running. Valid values include:

NO Do not check the new MCA request is from the same network address as the instance that is already running.

YES Check that the new MCA request is from the same network address as the instance that is already running.

The default is NO.

If the Adopt MCA Check parameter is set to (Y)es, the channel will only be adopted if the new MCA request is from the same network address as the instance that is already running.

Bullet-proof channels

WebSphere MQ channels over TCP/IP are difficult to handle when network failures occur. If the TCP/IP connection is broken when an WebSphere MQ channel is active, it is not at all uncommon for the receiving end of the channel to "hang" indefinitely in a TCP/IP receive call.

When connectivity is restored, the sender channel is generally unable to reconnect to the hanging receiver. In order to restart the channel, operator intervention is required to forcibly stop the channel. Once this is done, the sending side can normally reconnect.

The circumvention for this problem, on some MQ platforms, has been to use the TCP/IP KeepAlive function by adding a stanza to the qm.ini file (mqs.ini for clients) reading "TCP: KeepAlive=Yes". With this stanza in place, WebSphere MQ will enable the SO_KEEPALIVE option on the socket.

This results in TCP/IP itself sending packets across the link from time to time to verify the connection. If enough packets in a row are lost, the connection is presumed to be lost.

From an WebSphere MQ perspective, the receiving side of the channel is notified by TCP/IP that the connection is gone, and thus given a chance to shut down gracefully. Subsequent reconnection attempts by the sending side of the channel can then proceed normally without operator intervention.

TCP/IP KeepAlive is an excellent solution to this problem, but it has one significant drawback, that is, the KeepAlive timeout interval for connections is generally tunable only on a machine wide basis. In terms of WebSphere MQ channels, a timeout on the order of a few minutes might be reasonable. However, there may be other programs which rely on a timeout of one or two hours. If TCP/IP KeepAlive is the only solution, then WebSphere MQ may not coexist well with these other programs.

Rather than entering a potentially indefinite TCP/IP receive call, and relying on KeepAlive (if it has been properly configured and is in use) to wake up the channel, WebSphere MQ can instead enter a receive call for a finite amount of

time. At the end of this time, the queue manager has control to decide whether to receive again or to shut down the channel.

The facility to "wake up" channels waiting on a receive call has been named "bullet-proof channels".

Although it is the Receiver MCA that is generally waiting for data from the sender, during normal operation, the Sender MCA can be waiting for data from a receiver. In this case, following a communication failure, it is the Sender MCA that can remain in an indefinite wait state. Consequently, the bullet-proof channel feature applies to both sender and receiver channels.

Review section “Features” on page 15 for prerequisites for this feature.

Bullet-proof channel parameters

The bullet-proof channel feature is configurable on a per channel basis. The channel parameter that determines whether a channel will "wake up" after a configurable time is the Max TCP/IP Wait parameter.

The Max TCP/IP Wait parameter is configurable from the Channel Record screen, MQMT option 1.3.

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:12:29                    Channel Record           DISPLAY         CIC1
MQWMCHN                                                              A001
Channel  : MY.RCVR
 Desc. . :
 Protocol: T (L/T)  Type : R (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000000
 Get retry number . . . . . : 00000000   Short retry interval  . . : 000000000
 Get retry delay (secs) . . : 00000015   Long retry interval . . . : 000000000
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000000000
 Property control . . . . . : C
 Transmission queue name. . :
 TP name. . :
Sender/Receiver/Server/Requester
 Connection :
 Max Messages per Batch . . : 000050     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0040960    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 065535     Split Msg(Y/N)  . . . . . : Y
 Max TCP/IP Wait  . . . . . : 000300     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

Figure 21. Channel Record, bullet-proof channel parameter

The Max TCP/IP Wait parameter specifies a period of time (in seconds) for which the channel will wait to receive data from a remote sender. If no data is received within the specified period, the channel will terminate with an error.

By setting the Max TCP/IP Wait parameter to 0, the channel will wait indefinitely to receive data from a remote sender. Effectively, this disables the bullet-proof feature for the channel.

Care must be taken not to specify Max TCP/IP Wait value that is less than the disconnection interval for the channel. The disconnection interval is a parameter of

the sender channel definition, and determines how long the sender will keep a channel open when its transmission queue is empty. If the Max TCP/IP Wait value is less than the disconnection interval, the channel will always terminate with an error.

Note: For WebSphere MQ for z/VSE, the disconnection interval is equivalent to the Get retry number multiplied by the Get retry delay of the sender channel.

Chapter 4. System operation

There are six ways of managing an WebSphere MQ for z/VSE system:

- You can use the CICS transaction MQMT.

  MQMT allows you to configure, operate, and monitor an WebSphere MQ for z/VSE system. MQMT also supports the browsing of message queues and is described in this chapter.
- You can use the WebSphere MQ Command Line interface (MQCL).

  MQCL supports management of queues and channels, and is described in Chapter 5, "Utilities and interfaces," on page 169.
- You can use Programmable Command Format (PCF) messages, as described in Chapter 8, "Programmable system management," on page 221.
- You can use WebSphere MQ Commands (MQSC), as described in Chapter 9, "WebSphere MQ commands," on page 507.
- You can use the WebSphere MQ Explorer interface, which is available on the Windows and Linux (x86) platforms, as described in "Administration using the WebSphere MQ Explorer" on page 154.
- You can use a web browser to access the WebSphere MQ master terminal and associated CICS transactions using the WebSphere MQ for z/VSE CICS Web Support (CWS) feature. For more information, refer to "Administration via a web browser" on page 161.

WebSphere MQ master terminal displays

The MQMT menus and display screens are organized in an *informal* hierarchy as depicted in the following diagram. The hierarchy is informal in the sense that non-hierarchical paths between screens can be invoked by using the function keys. For improved legibility, the chart omits certain exit and return paths available from lower level screens.

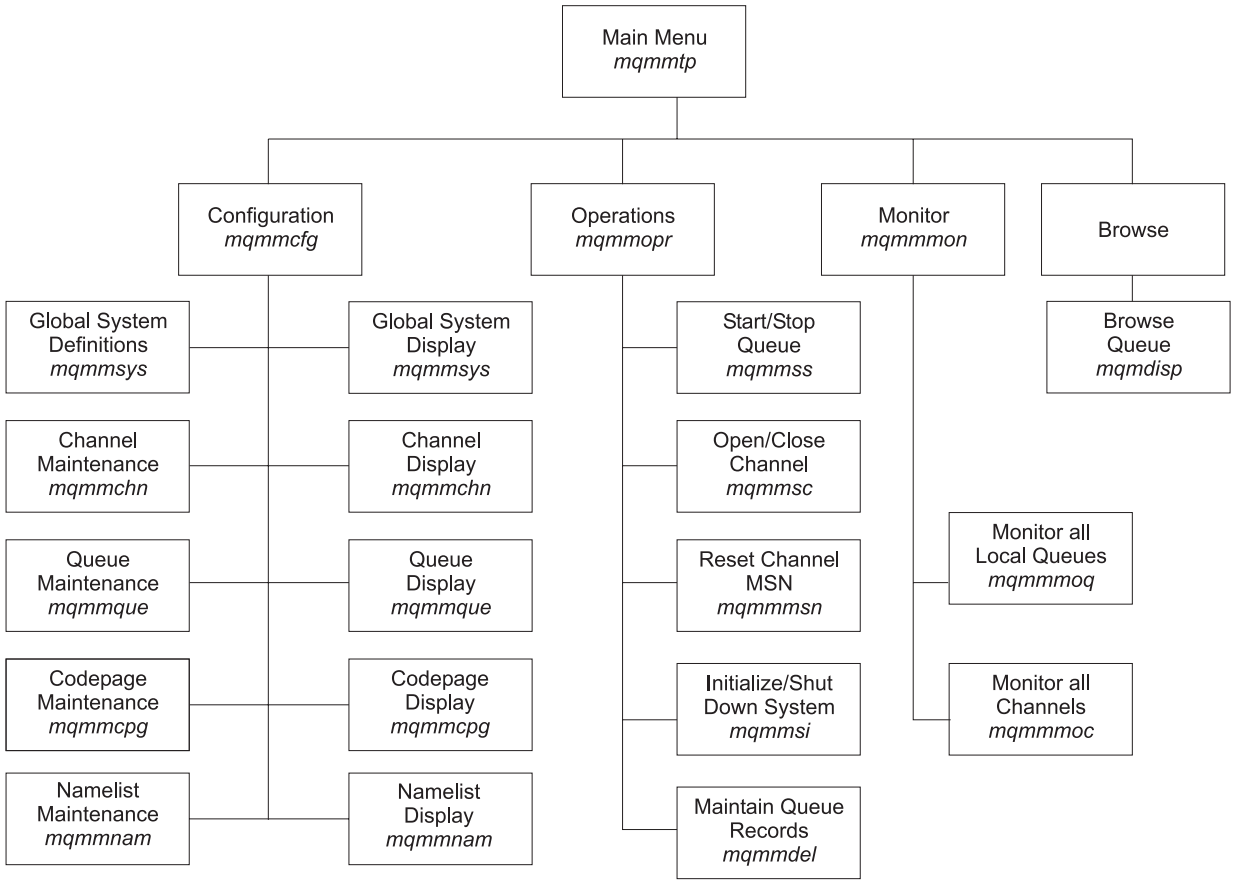


Figure 22. Display screen relationships

The main MQMT menu is shown in "WebSphere MQ master terminal (MQMT) – main menu" on page 79, and the operator functions available through each of the secondary panels are shown in "Configuration functions" on page 81.

General panel layout

WebSphere MQ panels are either menu panels or data entry panels. In either case, they show the following fields:

```
Date        IBM WebSphere MQ for z/VSE Version 3.0.0        CICS-Appl
Time                                                        Q-Manager
Panel-Id                                                    Termid









Message line

PF key line 1
PF key line 2 (if necessary)
```

Figure 23. General panel layout

Where:

CICS-Appl
The VTAM application ID for this CICS partition.

Panel-Id
The name of the displayed panel.

Q-Manager
The name of the WebSphere MQ queue manager specified in the global definitions.

Termid
The ID of the CICS terminal on which this panel is displayed.

WebSphere MQ master terminal (MQMT) – main menu

You can invoke the WebSphere MQ system administrator program, MQMT, from any 3270 terminal. To access the operator functions, type `MQMT` at the CICS prompt. The MQMT transaction cannot be invoked until the WebSphere MQ Setup Environment (MQSE) transaction has completed successfully.

When MQMT starts, the main menu is displayed.

```
07/06/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           MQBDTS
10:55:25           *** Master Terminal Main Menu ***                CIC1
MQMMTP                                                              A001
                          SYSTEM IS ACTIVE

                            1. Configuration

                            2. Operations

                            3. Monitoring

                            4. Browse Queue Records


                         Option:




Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
    CLEAR/PF3 = Exit                          ENTER=Select
```

Figure 24. Master terminal main menu

From the main menu, one of several submenus can be selected. The first three selections correspond to broad categories that include most WebSphere MQ operator functions:
- Configuring WebSphere MQ.
- Operating (controlling) WebSphere MQ.
- Monitoring WebSphere MQ.

The fourth function allows you to display the records on a selected queue:
- Browsing WebSphere MQ queues.

Each submenu presents a list of operator functions available from that screen. When a specific function is selected, the appropriate data entry or data display screens are presented to the operator.

Master Terminal transactions

The functions of the WebSphere MQ system administrator program can be invoked directly using the following transaction code table. For those customers using an External Security Manager, specific functions can be restricted to certain users or class of users. Alternatively, administration tasks can be restricted by enabling command and command resource security (for more information refer to “Resource definitions for command security” on page 666 and “Resource definitions for command resource security” on page 671).

```
MQMT  Master Terminal Main Menu
   |===> MQMC  Configuration Main Menu
   |        |===> MQMS  Global System Definition \
   |        |===> MQMQ  Queue Main Options        \maintenance
   |        |===> MQMH  Channel Record             \mode
   |        |===> MQMP  Code Page Definition        \
   |        |===> MQMN  Namelist Definition          \
|  |        |===> MQM1 Topic Definitions              \
|  |        |===> MQM2 Subscription Definitions        \
   |        |===> MQDS  Global System Definition \
   |        |===> MQDQ  Queue Main Options        \display
```

```
|       |===> MQDH  Channel Record             \mode
|       |===> MQDP  Code Page Definition        \
|       +===> MQDN Namelist Definition           \
|       |===> MQD1 Topic Definitions              \
|       +===> MQD2 Subscription Definitions        \
|
|===> MQMM  Monitor Main Menu
|       |===> MQQM  Monitor Queues
|       +===> MQCM  Monitor Channels
|
|===> MQMO  Operations Main Menu
|       |===> MQMA  Start / Stop Queue
|       |===> MQMB  Open / Close Channel
|       |===> MQMR  Reset Channel Message Sequence
|       |===> MQMI  Initialization / Shutdown of System
|       +===> MQMD  Maintain Queue Message Records
|
+===> MQBQ  Browse Queue Records
```

Operator screen action keys

The action keys available on each WebSphere MQ operator screen are displayed at the bottom of the screen with an explanation of their function. In general, the following keys are available and associated with the indicated action:

CLEAR - Exit MQMT
PF2 - Return to previous menu
PF3 - Exit to CICS
PF4 - Select/Read (Same as Return or Enter keys)
PF5 - Add
PF6 - Update
PF7 - Backward
PF8 - Forward
PF9 - Screen-dependent
PF10 - Screen-dependant
PF11 - Screen-dependant
PF12 - Delete/Screen-dependant

Configuration functions

Selecting option 1 (Configuration) from the master terminal main menu (see Figure 24 on page 80) displays the following screen:

```
2012/11/14      IBM WebSphere MQ for z/VSE Version 3.0.0            TSMQ300
15:30:25           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A004
                            SYSTEM IS ACTIVE

                Maintenance/Display Options
                          1 / 8     Global System Definitions
                          2 / 9     Queue Definitions
                          3 / 10    Channel Definitions
                          4 / 11    Code Page Definitions
                          5 / 12    Namelist Definitions
                          6 / 13    Topic Definitions
                          7 / 14    Subscription Definitions

                         Option: __




Please enter one of the options listed.
          5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Exit
```

Figure 25. Configuration Main Menu

On this screen, selections 1, 2, 3, 4 and 5 allow you to perform maintenance functions on various WebSphere MQ configuration objects. Selections 6, 7, 8, 9 and 10 allow viewing of the same objects.

Global system definition

Before you can do anything with messages and queues, you must configure a queue manager. Once the installation process is complete, you use the WebSphere MQ "Global System Definition" screen to configure the queue manager and start it.

Default objects form the basis of any object definitions that you make. System objects are required for queue manager operation and you must create these objects for the queue manager that you created.

Guidelines for configuring queue managers

A queue manager manages the resources associated with it, in particular the queues that it owns. It provides queuing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete WebSphere MQ objects. Some tasks you must consider when customizing a queue manager are:

- Selecting a unique queue manager name, as described by "Specifying a unique queue manager name."
- Creating the dead-letter and system log queues, as described by "Specifying the dead-letter and system log queues" on page 83.
- Backing up the configuration file, as described by "Backing up the configuration file after creating the queue manager" on page 96.

The tasks in this list are explained in the sections that follow.

Specifying a unique queue manager name: When you create a queue manager, you must ensure that no other queue manager has the same name, anywhere in

your network. Queue manager names are not checked at create time, and non-unique names will prevent you from using channels for distributed queuing.

One method of ensuring uniqueness is to prefix each queue manager name with its own (unique) node name. For example, if a node is called `accounts`, you could name your queue manager `accounts.saturn.queue.manager`, where `saturn` identifies a particular queue manager and `queue.manager` is an extension you can give to all queue managers. Alternatively, you can omit this, but note that `accounts.saturn` and `accounts.saturn.queue.manager` are *different* queue manager names.

If you are using WebSphere MQ for communicating with other enterprises, you can also include your own enterprise as a prefix. We do not actually do this in the examples, because it makes them more difficult to follow.

Specifying the dead-letter and system log queues: It is likely that the system log queue was created during installation, and the dead-letter queue was created using the MQJINSG.Z sample object definition job. For more details, see "Default object definitions" on page 37. If these queues have been created, skip this section.

The dead-letter queue is a local queue where messages are put if they cannot be routed to their correct destination.

Attention: It is vitally important to have a dead-letter queue on each queue manager in your network. Failure to do so may mean that errors in application programs cause channels to be closed or that replies to administration commands are not received.

You create a dead-letter queue as a local queue; "Creating local queues" on page 97 for details.

For example, if an application attempts to put a message on a queue on another queue manager, but the wrong queue name is given, the channel is stopped, and the message remains on the transmission queue. Other applications cannot then use this channel for their messages.

The channels are not affected if the queue managers have dead-letter queues. The undelivered message is simply put on the dead-letter queue at the receiving end, leaving the channel and its transmission queue available.

Therefore, when you create a queue manager you should specify the name of the dead-letter queue.

Similarly, the system log queue is essential for normal queue manager operation. The system log is used by the queue manager to report diagnostic and error messages. Some informational messages are generated when the queue manager is started, consequently, the system log queue should be defined to the queue manager before the system is started for the first time.

Like the system dead-letter queue, the system log is an WebSphere MQ queue and should be defined as a local queue.

Configuring the queue manager

For each installation of the WebSphere MQ system, one (and only one) queue manager must be defined. This is accomplished through the screen shown in Figure 26 on page 84. This screen is also used to modify the default or previously

defined global parameters.

```
2013/01/25        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
05:46:06                   Global System Definition                  CIC1
MQWMSYS                  Queue Manager Information                   A002
Queue Manager . . . . . .: PTHVSEC
Description Line 1. . . . .: ZVSE 4.3 SYSTEM ON PTHVSEC
Description Line 2. . . . .: ...
                           System Values
Maximum Connection Handles.: 00000100       System Wait Interval : 00000030
Maximum Concurrent Queues .: 00000100       Max. Recovery Tasks  : 0000
Allow TDQ Write on Errors  : Y   CSMT       Local Code Page  . . : 01047
Allow Internal Dump  . . . : Y              Subsystem id . . . . : MQV1
                                            Channel Auth Enabled : Y
                         Queue Maximum Values
Maximum Q Depth . . . . . .: 00010000       Maximum Global Locks.: 00001000
Maximum Message Size. . . .: 00409600       Maximum Local Locks .: 00001000
Maximum Single Q Access . .: 00000100       Max Properties Length: 00004094
                         Global QUEUE /File Names
Configuration File. : MQFCNFG
LOG Queue Name. . . : SYSTEM.LOG
Dead Letter Name. . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name. : SYSTEM.MONITOR
Requested record displayed.
PF2=Return PF3=Quit PF4/ENTER=Refresh    PF6=Update
                    PF9=Communications  PF10=Log    PF11=Events  PF12=Exits
```

Figure 26. System queue manager information

On this screen, the data entry fields are:

Queue Manager
This is the name of the local queue manager for this WebSphere MQ system installation. The name may be up to 48 characters and must conform to the WebSphere MQ naming requirements. For details, see "Object names" on page 3.

Description Lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Queue System Values:

Maximum Connection Handles
The maximum number of simultaneous connections to the queue manager.

Maximum Concurrent Queues
The maximum number of simultaneously open queues.

Allow TDQ Write on Errors
Y - allow writes to the CICS TDQ 'CSMT' if SYSTEM.LOG not available

N - do not allow write to the CICS TDQ

B - write to both SYSTEM.LOG and the CSMT TDQ.

System Wait Interval
The sleep time in seconds for the system monitor program and startup of trigger programs after system initialization. Thirty seconds is generally sufficient.

Max. Recovery Tasks
Maximum number of tasks attached by the system monitor when errors are detected in queues or control blocks attached to queues. A high number would lead to the use of too many CICS resources and have a negative impact on the overall CICS performance. The suggested value is zero.

Allow Internal Dump
Allow the WebSphere MQ API to process a CICS Task Dump if the internal areas are corrupted.

Local Code Page
The code page in use on the local system. If you plan to support remote client connections, you must use a local code page that can be translated into the code page of the remote client system. Generally, code page 1047 is a good choice, because many translations for this code page are provided with LE. Alternatively, you can define your own translation tables (see Appendix F, "WebSphere MQ server," on page 1009) and set the local code page appropriately.

Subsystem id
The queue manager's subsystem identifier. This attributes is only used by the security feature. It is a read-only attribute and can only be changed by changing the SYSIN.z installation file, rerunning the MQJSETUP.Z job followed by the MQSU transaction.

The default value for this attribute is MQV1.

Queue Maximum Values:

Maximum Q Depth
The maximum number of records that will be left unread on a queue.

Maximum Message Size
The maximum size of any message.

Maximum Single Q Access
The maximum number of object handles allowed for a queue.

Maximum Global Locks
The maximum number of entries that the queue manager uses to maintain destructive PUT or GET locks, per queue, for the system.

Maximum Local Locks
The maximum number of entries that an application can use to maintain destructive PUT, or GET locks, per queue, for each individual task.

Max Properties Length
The maximum length of property data in bytes that can be associated with a message.

Global QUEUE /File Names:

Configuration File
The CICS file definition name of the WebSphere MQ configuration file.

LOG Queue Name
The queue name where the WebSphere MQ programs write information and error messages. This is the system log queue.

Dead Letter Name
The file where channel programs write messages that are received with the wrong queue manager name or queue name. These messages will have the dead letter header placed in front of the queue record.

Monitor Queue Name
Diagnostic queue for MQI monitoring. The MQI monitor can be activated using MQMT option 2.1 (for more details refer to "Queuing System Request" on page 137.)

Note: Queue maximum value fields restrict the allowed values in the queue definition field values, while the rest of the fields affect the run-time values when the System is initialized.

Queue Manager Communications Settings: Press PF9 (Comms) on the Global System Definition screen to display the Queue Manager Communications Settings

screen:

```
2011/10/31        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
21:08:43                  Global System Definition                  CIC1
MQWMSYS                    Communications Settings                  A000

    TCP/IP settings                          Batch Interface settings
    Licensed clients . . : 00000             Batch Int. identifier: MQBSRV39
    Adopt MCA  . . . . . : N                 Batch Int. auto-start: Y
    Adopt MCA Check  . . : N
                                             Channel Auto-Definition
                                             Auto-definition  . . : N
    SSL parameters                           Auto-definition exit :
    Key-ring sublibrary  :
    Key-ring member  . . :
    SSL reset count  . . :

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : Y

Requested record displayed.
PF2=Return PF3=Quit PF4/Enter=Read PF6=Update PF10=Listeners PF11=Services
```

Figure 27. Queue manager communications settings

TCP/IP settings:

Licensed clients
The number of clients for which the WebSphere MQ system is registered. This represents the maximum number of concurrent remote client connections that the local system will support at any one time. Use the number from your WebSphere MQ for z/VSE license documentation.

Adopt MCA
Indicates whether an Message Channel Agent (MCA) should adopt another MCA process if one is already running. For more information about the Adopt MCA feature, see "Adopt MCA" on page 72.

Adopt MCA Check
Indicates whether the network address of an existing MCA should be checked before adopting the MCA process. For more information, see "Adopt MCA" on page 72.

SSL parameters:

Key-ring sublibrary
The key-ring sublibrary identifies the z/VSE sublibrary name that contains the private key and certificate intended for use by SSL enabled MQ channels. This is the SSL product KEYLIB or key ring file name. Queue managers that require SSL enabled channels, must identify a valid key-ring sublibrary.

Key-ring member
The key-ring member is the key-ring sublibrary member name of the SSL private key (.PRVK) and certificate (.CERT) files. Queue managers that require SSL enabled channels, must identify a valid key-ring member name.

SSL reset count
Specifies when SSL channel MCAs that initiate communication reset the secret key used for encryption on the channel. The value of this parameter represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. This number of bytes includes control information sent by the MCA. The secret key is renegotiated when (whichever occurs first):

- The total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or
- If channel heartbeats are enabled, before data is sent or received following a channel heartbeat.

Specify a value in the range zero through 999 999 999. A value of zero, the queue manager's initial default value, signifies that secret keys are never renegotiated.

For more information regarding SSL features, refer to Chapter 11, "Secure Sockets Layer services," on page 643.

PCF parameters:

System command queue
The command queue where local and remote administration applications can place PCF messages to be processed by the local queue manager. The default value for this parameter is: SYSTEM.ADMIN.COMMAND.QUEUE.

For more information, refer to Chapter 8, "Programmable system management," on page 221.

System reply queue
The command reply queue used by the MQSC batch utility program. For more information, refer to Chapter 9, "WebSphere MQ commands," on page 507.

Cmd Server auto-start
Indicator for the automatic activation of the PCF command server. This parameter can have the following values:

Y Automatically start the PCF command server when the queue manager is initialized.

N Do not automatically start the PCF command server when the queue manager is initialized.

Cmd Server convert
Indicator for the data conversion of PCF messages by the command server. This parameter can have the following values:

Y Apply data conversion to PCF messages retrieved by the command server.

N Do not apply data conversion to PCF messages retrieved by the command server.

Cmd Server DLQ store
Indicator for the storage of undeliverable PCF reply messages to the system dead letter queue by the command server. This parameter can have the following values:

Y Command server will attempt to place undeliverable PCF response messages to the system dead letter queue.

N Command server will not attempt to place undeliverable PCF response messages to the system dead letter queue.

Batch Interface settings:

Batch Int. identifier
Batch interface identifier. This is a 1-8 character identifier used by batch MQ applications to connect to a relevant queue manager. The identifier must be unique within the context of a z/VSE system. The default value for this parameter is MQBISERV.

For more information refer to "Using the batch interface" on page 178.

Batch Int. auto-start
Indicator for the automatic activation of the batch interface. This parameter can have the following values:

Y Automatically start the batch interface when the queue manager is initialized (and stop the batch interface when the queue manager is stopped).

N Do not automatically start the batch interface when the queue manager is initialized.

Channel Auto-Definition settings::

Auto-definition
Switch for the channel auto-definition feature. This parameter can have the following values:

Y Channel auto-definition enabled.

N Channel auto-definition disabled.

Auto-definition exit
Channel auto-definition exit name. This is a 1-8 character program name that is called by the queue manager if the channel auto-definition feature is enabled. This field is optional. If used, it must name a user-written program defined to your CICS system.

Queue Manager Log and Trace Settings: Press PF10 (Log) on the Global System Definition screen to display the Queue Manager Log and Trace Settings screen:

```
12/03/2010        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:42:46                  Global System Definition                  CIC1
MQWMSYS                    Log and Trace Settings                   A000

    Log Settings             Q C       Accounting monitoring
      Informational . . . : Y N          MQI accounting . . . . . : N
      Warning . . . . . . : Y N          Queue accounting . . . . : N
      Error . . . . . . . : Y N          Acc. Conn override . . . : N
      Critical  . . . . . : Y N          Accounting interval  . . : 001800
      Communication . . . : Y N        Statistics monitoring
      Reorganization  . . : Y N          MQI statistics . . . . . : N
      System  . . . . . . : Y N          Queue statistics . . . . : N
                                         Channel statistics . . . : N
    Trace Settings                       Statistics interval  . . : 001800
      MQI calls . . . . . : Y          Online monitoring
      Communication . . . : Y            Queue monitoring . . . . : N
      Reorganization  . . : Y            Channel monitoring . . . : N
      Data conversion . . : Y          Recording
      System  . . . . . . : Y            Activity . . . . . . . . : M
                                         Trace Route  . . . . . . : M

Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 28. Queue manager log and trace settings

Logging, in this sense, refers to the type or severity of messages written to the SYSTEM.LOG. Tracing refers to entries written to the CICS auxiliary trace. Configurability is intended to reduce certain processing overhead involved with logging and tracing under WebSphere MQ for z/VSE. It is expected that many customers, in a production environment, will reduce logging to error and critical messages only, and switch tracing off altogether. You can also view log and trace settings from MQMT option 1.5.

Log Settings: Log settings involve a choice between logging by the severity of messages and/or the general type of message. For example, it is possible to select logging for error and critical messages only, along with, for example, general system messages. This is possible with the following log settings:
Informational
N
Warning
N
Error Y
Critical
Y

and/or
Communication
N
Reorganization
N
System
Y

With this configuration, all general system messages would be written to the SYSTEM.LOG (including informational and warning messages), otherwise, only error and warning messages would be written to the log.

Log settings are made under the column labeled "Q" (for Queue). Valid values include:

N Suppress messages of this severity/type.

Y Send messages of this severity/type to the system log queue.

Diagnostic and error messages can optionally be sent to the z/VSE console. A message cannot be sent to the console unless it is also sent to the system log queue. Consequently, for example, it is not possible to suppress informational messages and also have them sent to the z/VSE console.

Settings for optional logging to console are made under the column labeled "C" (for Console). Valid values include:

N Do not send messages of this severity to the console.

Y Send messages of this severity to the console.

R Send messages of this severity to the console and prompt for an operator reply.

Messages sent to the console are prefixed with the generic WebSphere MQ message identifier MQI0200I, followed by the message identifier and text of the message written to the system log queue. The MQI0200I message is truncated to a single console line if necessary.

Messages sent to the console requiring an operator reply are highlighted and remain on the z/VSE console until an operator reply is registered, or the system OPERTIM expires. The text “...awaiting reply” is appended to messages sent to the console that require an operator response.

Care should be taken not to flood the z/VSE console with messages (particularly messages requiring an operator response). To avoid flooding the console, it is recommended that a setting of "R" (for Reply) is only used for Critical messages.

Trace settings: Trace settings involve selection by general type. For example, it is possible to trace communications programs and general system programs, and exclude tracing for MQI calls, reorganization and data conversion. This example is possible with the following trace settings:

MQI calls
N
Communication
Y
Reorganization
N
Data conversion
N
System
Y

Normally, tracing is only required when a serious system problem has been encountered, and IBM service personnel have requested a trace of MQ system activity. Since tracing involves some system overhead, it is recommended that during normal operation, tracing is deactivated (that is, set all selections to “N”).

Accounting monitoring: These settings control if and how accounting information is collected. The following settings are available:

MQI Accounting
Controls whether accounting information for MQI data is to be collected. Valid values are:
Y MQI accounting enabled.
N MQI accounting disabled.

Queue accounting
Controls the collection of accounting data for queues. Valid values are:
X Disable queue accounting for all queues.
Y Queue accounting enabled.
N Queue accounting disabled.

Acc. Conn override
Specifies whether applications can override the settings of the Queue accounting and MQI accounting queue manager parameters. Valid values are:
Y Override enabled.
N Override disabled.

Accounting interval
The time interval, in seconds, at which intermediate accounting records are written. Specify a value in the range 1 through 604,000.

Statistics monitoring: These settings control if and how statistics information is collected. The following settings are available:

MQI statistics
Controls whether statistics information for MQI data is to be collected. Valid values are:
Y MQI statistics enabled.
N MQI statistics disabled.

Queue statistics
Controls the collection of statistics data for queues. Valid values are:
X Disable queue statistics for all queues.
Y Queue statistics enabled.
N Queue statistics disabled.

Channel statistics
Controls the collection of statistics data for channels. Valid values are:
X Disable channel statistics for all channels.
N Disable channel statistics for those channels that default to the queue manager's channel statistics setting.
L Collect a low-level of channel statistics information.
M Collect a medium-level of channel statistics information.
H Collect a high-level of channel statistics information.

Statistics interval
The time interval, in seconds, at which intermediate statistics records are written. Specify a value in the range 1 through 604,000.

Online monitoring: These settings control if and how real-time monitoring information is collected. The following settings are available:

Queue monitoring
Default setting for online monitoring for queues. Valid values are:
X Disable queue monitoring for all queues.
N Disable queue monitoring for those queues that default to the queue manager's queue monitoring setting.
L Collect a low-level of queue monitoring information.

M Collect a medium-level of queue monitoring information.
H Collect a high-level of queue monitoring information.

Channel monitoring
Default setting for online monitoring for channels. Valid values are:
X Disable channel monitoring for all channels.
N Disable channel monitoring for those channels that default to the queue manager's channel monitoring setting.
L Collect a low-level of channel monitoring information.
M Collect a medium-level of channel monitoring information.
H Collect a high-level of channel monitoring information.

Recording Activity
M 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.
Q The queue manager is enabled for activity recording. Any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE.
D The queue manager is disabled for activity recording. No activity reports are generated in this queue manager.

Recording Trace route
M The queue manager is enabled for trace-route messaging. Applications can write activity information to the trace-route message.
Q The queue manager is enabled for trace-route messaging. Applications can write activity information to the trace-route message delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.
D 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.

Queue Manager Event Settings: Press PF11 (Event) on the Global System Definition screen to display the Queue Manager Event Settings screen:

```
12/03/2010        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:11:09                  Global System Definition                  CIC1
MQWMSYS                        Event Settings                       A000

    Event queues
    Queue manager events : SYSTEM.ADMIN.QMGR.EVENT
    Channel events . . . : SYSTEM.ADMIN.CHANNEL.EVENT
    Performance events . : SYSTEM.ADMIN.PERFM.EVENT
    Command events . . . : SYSTEM.ADMIN.COMMAND.EVENT
    Configuration events : SYSTEM.ADMIN.CONFIG.EVENT

    Qmgr events         Channel events      Performance events
    Inhibit  . . . : N  Started  . . . : N  Queue depth  . . : N
    Local  . . . . : N  Stopped  . . . : N  Service interval : N
    Remote . . . . : N  Conversion err : N
    Authority  . . : N  Auto-define. . : N
    Start/Stop . . : Y  SSL  . . . . . : N
    Command  . . . : N
    Configuration  : N


Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 29. Queue Manager event settings

Event queues:

Queue manager events, event queues
The queue manager events queue names the target queue for queue manager event messages. The name specified should be a valid local queue. The MQJQUEUE.Z files contains sample VSAM definitions for files to host the event queues. The default name for the queue manager event queue is SYSTEM.ADMIN.QMGR.EVENT, and the default VSAM file for this queue is MQFIEQE.

Channel events
The channel events queue names the target queue for channel event messages. The name specified should be a valid local queue. The MQJQUEUE.Z files contains sample VSAM definitions for files to host the event queues. The default name for the channel event queue is SYSTEM.ADMIN.CHANNEL.EVENT, and the default VSAM file for this queue is MQFIECE.

Performance events
The performance events queue names the target queue for performance event messages. The name specified should be a valid local queue. The MQJQUEUE.Z files contains sample VSAM definitions for files to host the event queues. The default name for the performance event queue is SYSTEM.ADMIN.PERFM.EVENT, and the default VSAM file for this queue is MQFIEPE.

Command events
The command events queue names the target queue for command event messages. The name specified should be a valid local queue. The MQJQUEUE.Z files contain sample VSAM definitions for files to host the event queues. The name for the command event queue is SYSTEM.ADMIN.COMMAND.EVENT and is not configurable, and the default VSAM file for this queue is MQFIEME.

Configuration events
The command events queue names the target queue for configuration

event messages. The name specified should be a valid local queue. The MQJQUEUE.Z files contain sample VSAM definitions for files to host the event queues. The name for the configuration event queue is SYSTEM.ADMIN.CONFIGURATION.EVENT and is not configurable, and the default VSAM file for this queue is MQFIENE.

Qmgr events:

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 respectively.

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, when an application attempts to access an object that has not been defined.

Remote
Remote events indicate that an application (or the queue manager) cannot access a (remote) queue on another queue manager. For example, when the transmission queue to be used is not correctly defined.

Authority
Authority events indicate that an authorization violation has been detected. For example, an application attempts to open a queue for which it does not have the required authority, or a command is issued from a user ID that does not have the required authority.

Start/Stop
Start and stop events indicate that a queue manager has been started or has been requested to stop

Command
The Command attribute accepts these values:
N Disabled.
Y Enabled.
X No Display. Command events are generated except for PCF Inquire and MQSC DISPLAY commands.

Configuration
The Configuration attribute accepts these values:
N Disabled.
Y Enabled.

Channel events:

Started
Channel started events are generated when a Sender or Receiver channel starts. Channel started events are not generated for Client (SVRCONN) channels.

Stopped
Channel stopped events are generated when a Sender or Receiver channel stops. Channel stopped events are not generated for Client (SVRCONN) channels.

Conversion err
Channel conversion error events are generated by Sender channels when an MQGET call to the transmission queue fails with a data conversion error.

Auto-define
Channel auto-definition events are generated by the Receiver MCA when the auto-definition feature is enabled and a receiver or server connection channel is automatically defined.

SSL SSL error events are generated when an SSL-enabled channel encounters an SSL error.

The SSL attribute accepts these values:
N Disabled.
Y Enabled.

Performance events:

Queue depth
Queue depth events are related to the queue depth, that is, the number of messages on the queue. The types of queue depth events are: Queue Depth High, Queue Depth Low and Queue Full.

Service interval
Queue service interval events indicate whether a queue was "serviced" within a user-defined time interval called the service interval. Depending on the circumstances, queue service interval events can be used to monitor whether messages are being taken off queues quickly enough.

Queue Manager API Exit Settings: Press PF12 (Ext) on the Global System Definition screen to display the Queue Manager MQ API Settings screen:

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:51:25                  Global System Definition                  CIC1
MQWMSYS                       MQ API Settings                       A000

   Local API Exits
   1. Name: MQ_SAMPLE_EXIT
      Module: MQPSAXE     Data:
   2. Name:
      Module:             Data:
   3. Name:
      Module:             Data:
   4. Name:
      Module:             Data:
   5. Name:
      Module:             Data:
   6. Name:
      Module:             Data:
   7. Name:
      Module:             Data:
   8. Name:
      Module:             Data:
Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 30. Queue manager MQ API settings

Local API Exits: You can configure up to 8 MQ API exits and associated exit data. MQ API exit programs are called in the sequence listed.

Name The name of the MQ API exit is an MQ object name, not the name of the exit program itself. This can be from 1 to 48 characters and must follow the naming standards for all WMQ objects. See "Object names" on page 3.

Module
The name of the MQ API exit program. MQ API exit names can be 1-8 characters, and must follow the naming standards for all programs defined in the CICS CSD.

Data The exit data passed to the MQ Exit module on invocation. Exit data can be from 0 to 32 characters in length.

For more information about MQ API exits, refer to Chapter 13, "API exits," on page 677.

Backing up the configuration file after creating the queue manager

When you create the queue manager, the queue manager configuration file MQFCNFG is updated. This contains configuration parameters for the queue manager, and the queue and channel definitions.

You should make a backup of this file. If you have to create another queue manager, perhaps to replace the existing queue manager if it is causing problems, you can reinstate the backup when you have removed the source of the problem.

If you restore the configuration file then be aware that you will reset all channel MSNs to the values they held at the time of the backup. You may need to reset the channel MSNs using the master terminal transaction or by using batch utility MQPUTIL. You will also lose all queue and channel definitions made to the queue manager since the MQFCNFG file was backed up. For this reason, the backup should be retaken after new definitions have been introduced and deemed stable.

Queue definitions

Selecting 2 on the Configuration menu allows you to maintain (add, modify, or delete) queue definitions for the local installation of WebSphere MQ.

Note: The same screens are used to accomplish the three functions of adding, modifying, or deleting a queue definition; the required action being selected through the function keys. The following sections present the screens that you see if you are adding a new queue definition.

"Modifying and deleting queue definitions" on page 110 explains how you modify or delete a queue.

To create a queue definition, multiple screens may be involved. The first screen is the same for all queues, and allows entry of the queue name and type.

Based on the type you enter, an appropriate second screen is displayed for you to enter the remainder of the data to complete the definition.

In the case of local queues, a third screen, the Extended Queue Definition Screen can be displayed.

The first screen displayed is as shown in Figure 31 on page 97.

```
12/24/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:34:45                Queue Main Options                           CIC1
MQWMQUE                                                              A001
                              SYSTEM IS ACTIVE

       Default Q Manager. : VSE.TS.QM1

       Object Type. . . . :        L = Local Queue
                                   M = Model Queue
                                   R = Remote Queue
                                   AQ = Alias Queue
                                   AM = Alias Queue Manager
                                   AR = Alias Reply Queue

       Object Name. . . . :









PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

Figure 31. Queue main options screen

On this screen, the data entry fields are:

Object Type
This is a two character field with the acceptable entries listed on the screen. The type determines the next screen to be displayed.

Object Name
This is the name of the queue (or alias) being defined. The name may be up to 48 characters, must be unique among all other defined queues for this installation, and must conform to the WebSphere MQ naming requirements.

The Object Type you select on this screen is used to determine which of the definition screens is displayed:

- L selects a local queue definition; see "Creating local queues"
- M selects a model queue definition; see "Create model queue" on page 104
- R selects a remote queue definition; see "Create remote queue" on page 106
- AQ selects an alias queue definition; see "Create alias queue" on page 108
- AM selects an alias manager definition; see "Create alias queue manager" on page 108
- AR selects an alias reply queue definition; see "Creating alias reply queues" on page 109

Creating local queues

Selecting "L" on the screen in Figure 31 displays the screen shown in Figure 32 on page 98.

```
11/17/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:17:42               Queue Definition Record                      CIC1
MQWMQUE                 Local Queue Definition                      A000

Object Name  . . . . . . . : ANYQ
Description line 1 . . . . : Test queue
Description line 2 . . . . :

Dual Update Queue  . . . . :

Put Enabled  . . . . . . . : Y    Inbound status . . : A
Get Enabled  . . . . . . . : Y    Outbound status  . : A

Accounting, Statistics & Monitoring
Queue accounting . . . . . : Q    Queue monitoring . : Q
Queue statistics . . . . . : Q

Automatic Reorganization
Reorganize . . . . . . . . : N    Start Time: 0000 Interval: 0000
VSAM Catalog . . . . . . . :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF10=Queue  PF12=Delete
```

Figure 32. Local queue definition

On this screen, the data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Put Enabled
A toggle that enables or disables MQPUT operations against this queue.

Get Enabled
A toggle that enables or disables MQGET operations against this queue.

Inbound status
Sets the initial status to active (A) or inactive (I) at run time for the inbound direction of the queue.

Outbound status
Sets the initial status at run time for the outbound direction of the queue.

Dual Update Queue
When an existing queue name is entered here, dual queuing is activated. The queue being created becomes the primary queue, and the queue entered in this field becomes the dual queue. The definition of the dual queue is updated automatically with the name of the primary queue. The queue display of the dual queue has a corresponding heading “Dual Source Queue”.

Dual Source Queue
The name of the primary queue, for which the queue being displayed is the dual queue. This field appears only when a local queue serves as a dual update queue.

Note: When an existing queue is defined as the dual to a primary queue, these two queues both participate in the same logical units of work.

If, for any reason, it becomes impossible to update the dual queue (for example, if the queue becomes disabled, the associated file is closed, or an ISC link is lost), updates continue to be made to the primary queue and the dual queue goes to a recovery status.

Queue accounting
The collection of queue accounting information is controlled by this attribute and the queue manager attribute of the same name.

Queue statistics
The collection of queue statistics information is controlled by this attribute and the queue manager attribute of the same name.

Queue monitoring
The collection of queue monitoring information is controlled by this attribute and the queue manager attribute of the same name.

Automatic Reorganize
A toggle that enables or disables automatic reorganization of the VSAM file associated with the selected queue, at specified time intervals.

Note:

1. WebSphere MQ for z/VSE uses VSAM files to store messages. The indexes of these files can become fragmented, causing the performance of the system to suffer.

   To reorganize these files you must use VSAM utilities, and the Automatic Reorganize feature automates the process.

   If automatic reorganization is not suitable then use the batch reorganization utility MQPREORG.
2. You are recommended to use the Automatic Reorganize feature only for queue files with high activity.
3. When you specify a queue file that is to be automatically reorganized, you should ensure that there is only one WebSphere MQ queue associated with each VSAM file.
4. The Start Time identifies the time of day when the reorganization runs. For example, 0230 is 2:30 a.m., and 2345 is 11:45 p.m..
5. The Interval, in minutes, identifies the frequency that the reorganization runs. For example, 1440 is daily, and 0120 is every two hours.

   If zero is specified, then the reorganization will only be scheduled once per WebSphere MQ for z/VSE start up. If WebSphere MQ for z/VSE is run longer than 24 hours, then specify 1440 to start reorganization every day at specified start time.
6. The VSAM Catalog field is now redundant and any data entered is treated as a comment. Likewise any data displayed is not to be regarded as meaningful.
7. The reorganization can take place even when there are messages on the queue. Message data is retained. However, logically deleted messages are removed.
8. Be aware that during reorganization, application programs do not have access to the queue. If there are any active applications, the reorganization will retry or postpone. Applications that attempt to access the queue after the reorganization starts will wait. The processing time for the reorganization varies, depending on the number of unprocessed messages remaining on the queue.

9. If specifying automatic reorganization for a queue then it is required that the VSAM file containing the queue be defined with cluster data name with suffix of .DATA and index name with suffix of .INDEX. This is also required for the MQFREOR file. And again, both the queue file and MQFREOR must be defined in the same VSAM catalog.
10. An automatic reorganization is scheduled during WebSphere MQ for z/VSE initialization for all local queues which have reorganization enabled. Also, when a local queue is added with reorganization enabled, or an existing local queue is enabled for reorganization, or is currently enabled and the start time or interval is modified, the reorganization is scheduled or rescheduled.
11. When the queue manager is started, or a queue's interval value is changed, the queue manager schedules the next reorganization. The queue manager uses the start time and interval values relative to the current time to determine when a reorganization will be scheduled. For example, if the queue manager is started at 09:25am:

```
Start time   Internal    Scheduled
 0100         0000         01.00
 0115         0120         11.15
 2300         0000         23.15
 2315         0720         11.15
 0745         0060         09.45
```

12. Changing the reorganization interval, when reorganization is enabled, will result in scheduling a reorganization using the new interval.
13. Reorganizations are serialized, as there is only one MQFREOR file. If a large number of reorganizations are scheduled for the same time, and a queue reorganization cannot start within 10 minutes, then the reorganization is rescheduled for the next interval.

Local queue extended definition screen: By pressing function key 10 (PF10), you can display a second screen to enter the extended definition fields for the queue. In the case of a request to add a queue, this Extended Definition Screen is presented automatically. This detailed screen is shown in Figure 33 on page 101:

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
06:55:56               Queue Extended Definition                    CIC1
MQWMQUE                                                             A001

Object Name: MY.LOCALQ

General               Maximums                   Events
Type   . . : Local    Max. Q depth . : 00010000  Service int. event: N
File name  : MQFI001  Max. msg length: 00040960  Service interval  : 00000000
Usage  . . : N        Max. Q users . : 00000100  Max. depth event  : N
Shareable  : Y        Max. gbl locks : 00000100  High depth event  : N
Dist.Lists : N        Max. lcl locks : 00000100  High depth limit  : 000
PropCtl. . : C                                   Low depth event . : N
Triggering                                       Low depth limit . : 000
Enabled  . : N        Transaction id.:
Type . . . :          Program id . . :
Max. starts: 0001     Terminal id  . :
Restart  . : N        Channel name . :
User data  :
           :

Record updated OK.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
            PF9=List  PF10=Queue
```

Figure 33. Local queue extended definition

Note:

1. The PF10 key can be used to toggle between the Local Queue Definition screen (Figure 32 on page 98) and the Local Queue Extended Definition screen (Figure 33).
2. Either Trans ID or Program ID is required if triggering is enabled.
3. The internal WebSphere MQ trigger API transaction MQ02 cannot be used as a trigger transaction ID.
4. Both a trigger transaction and a trigger program can be defined, but only the trigger transaction is activated and the trigger program name is passed in the trigger communications area.
5. The queue maximums are restricted by the queue manager's global maximums, and the maximum limits of the product.

Navigation through the screens is dependent upon the PF keys.

On this screen, the data entry fields are:

Object Name
Filled in from the previous screen. Cannot be modified.

Physical Queue Information:

Usage Normal (N) means that the queue is used by an application to receive inbound messages. Transmission (T) means that the queue is used by WebSphere MQ to hold outbound messages destined for another WebSphere MQ queue manager.

Shareable
Defines a queue as shareable or exclusive on input.

File name
The CICS file name, with a maximum of seven characters, used to store messages for this queue.

Note: The File name cannot be changed while there are active messages in the queue. A value of MQFREOR or MQFCNFG cannot be used as a file name.

Maximum Values:

Max. Q depth
The maximum number of messages allowed on this queue. The default value is the value specified in the global system definition.

Max. msg length
The maximum length of an application message processed on this queue.

Max. Q users
The maximum number of MQOPEN calls that can occur for this queue simultaneously.

Max. gbl locks
Allocates the locking table for this queue for all committed MQGET calls.

Max. lcl locks
This is used to allocate the locking table for this queue for each task's noncommitted MQGET calls.

Trigger Information:

Enabled
If you are defining a transmission queue for use with a sender channel, set this value to Yes (Y). Otherwise, for use with a server or receiver channel, set this field to No (N).

Alternatively, for queues with a usage of (N)ormal, set this parameter to (Y)es to instruct the queue manager to automatically start a trigger process instance in response to message activity on the queue.

If no trigger process is required, set this parameter to (N)o.

Type

F A trigger is generated when the queue goes from an empty to a non-empty state.

E A trigger instance is generated every time a message is placed on the queue. This can be constrained by the "Maximum Trigger Starts" parameter which sets a maximum number of trigger instances associated with the queue that can be running at any one time.

Only one transaction can be active against the queue if the Trigger Type is set to F.

Max. starts
The maximum number of trigger threads that can be active at once.

Restart
Indicates that the system monitor task (MQSM) should restart a trigger instance if it detects messages on the queue but no active trigger instance.

Transaction id
The name of the transaction to be started by a trigger, with a length of four characters. If a transaction ID is specified, this transaction will be started. For a transmission queue, this field is left blank.

If a transaction identifier is defined, the program identifier should be left blank.

Once the initial maximum trigger starts is reached then WebSphere MQ for z/VSE only checks that the maximum trigger starts are running at every system interval and not when each task completes. If it is important to have a definite number of trigger instances running against a queue, use Program ID to identify your trigger program.

Program id
The name of the user program to be invoked, with a length of eight characters. If you are defining a transmission queue to be used with a sender channel, MQPSEND must be used. If the field for Trans ID is left blank and this field contains a program ID, the specified program is linked.

Terminal id
Optional field of four characters used for problem determination. It is attached to the transaction ID specified in the Trans ID field.

Channel Name
Is the channel name, with a maximum of 20 characters.

This parameter is relevant for transmission queues only, and is used by the WebSphere MQ Sender MCA (MQPSEND) to identify the appropriate channel for transmitting messages to a remote WebSphere MQ system.

User Data
A field for static data to be passed to the trigger instance. When a trigger instance is activated, it is passed data in the form of the MQTM structure (see the CMQTML and CMQTMV copybooks). Data in the User Data field is passed in the MQTM-USERDATA field of the MQTM structure. The trigger instance can use this data for its own internal logic.

Event Information:

Service int. event
Specifies whether service interval events are required for this particular queue. Queue service interval events indicate whether a queue was 'serviced' within a user-defined time interval called the service interval. Depending on the circumstances at your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

There are two types of queue service interval event:

A Queue Service Interval OK event, which indicates that, following an MQPUT call or an MQGET call that leaves a non-empty queue, an MQPUT call or an MQGET call was performed within a user-defined time period, known as the service interval.

A Queue Service Interval High event, which indicates that, following an MQGET or MQPUT call that leaves a non-empty queue, an MQGET call was not performed within the user-defined service interval. This event message can be caused by an MQPUT call or an MQGET call.

To enable both Queue Service Interval OK and Queue Service Interval High events you need to 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.

These events are mutually exclusive, which means that if one is enabled the other is disabled. However, both events can be simultaneously disabled.

Valid values for this parameter include:
H - High
O - OK
N - None

Service interval
Specifies the interval used to determine whether the performance events High and Ok are generated. The value specified represents milliseconds.

Max. depth event
Indicates whether or not queue full events are required. A queue full event is generated when an attempt to put a message to the queue fails because the queue has reached its maximum queue depth.

High depth event
Indicates whether or not Queue Depth High events are required. A Queue Depth High event is generated when a message is put to the queue which increases the queue depth to the limit specified by the High depth limit parameter.

High depth limit
Specifies a queue depth as a percentage that is to be used to determine whether or not a Queue Depth High event should be generated. For example, a value of 80 for a queue with a maximum queue depth of 1000, would mean a Queue Depth High is generated when the queue depth reaches 800 messages.

Low depth event
Indicates whether or not Queue Depth Low events are required. A Queue Depth Low event is generated when a message is retrieved from the queue which decreases the queue depth to the limit specified by the Low depth limit parameter.

Low depth limit
Specifies a queue depth as a percentage that is to be used to determine whether or not a Queue Depth Low event should be generated. For example, a value of 20 for a queue with a maximum queue depth of 1000, would mean a Queue Depth Low is generated when the queue depth reduces to 200 messages.

Create model queue

Selecting “M” on the screen in Figure 31 on page 97, providing an object name for the model queue, and pressing PF5 to add a queue, displays the screen shown in Figure 34 on page 105.

```
11/17/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:25:05                Queue Definition Record                     CIC1
MQWMQUE                  Model Queue Definition                     A000



Object Name. . . . . . . . : REPLY.MODEL
Description line 1 . . . . :
Description line 2 . . . . :

Put Enabled  . . . . . . . : Y   Y=Yes, N=No
Get Enabled  . . . . . . . : Y   Y=Yes, N=No

Accounting, Statistics & Monitoring
Queue accounting . . . . :   Q   Queue monitoring . : Q
Queue statistics . . . . :   Q




Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF10=Queue  PF12=Delete
```

Figure 34. Model queue definition

On this screen, the data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 and 2
Text fields for operator use only. They may each be up to 32 characters.

Put Enabled
A toggle that enables or disables MQPUT operations against this queue.

Get Enabled
A toggle that enables or disables MQGET operations against this queue.

Queue accounting
The collection of queue accounting information is controlled by this attribute and the queue manager attribute of the same name.

Queue statistics
The collection of queue statistics information is controlled by this attribute and the queue manager attribute of the same name.

Queue monitoring
The collection of queue monitoring information is controlled by this attribute and the queue manager attribute of the same name.

Model queues are not used to store messages. Instead, they are used as a template when creating a dynamic queue. When an application opens a model queue, the queue manager dynamically creates a local queue with the attributes of the model queue.

Model queue extended definition: After setting general attributes for the model queue, the extended definition can be accessed by pressing PF5 (Add). This displays the screen shown in Figure 35 on page 106.

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:13:30               Queue Extended Definition                    CIC1
MQWMQUE                                                             A001

Object Name: A.MODEL.PERM

General               Maximums                   Events
Type   . . : Model    Max. Q depth . : 00100000  Service int. event: N
File name  : MQFI001  Max. msg length: 04000000  Service interval  : 00000000
Usage  . . : N        Max. Q users . : 00000100  Max. depth event  : N
Shareable  : Y        Max. gbl locks : 00000100  High depth event  : N
Def. type  : P        Max. lcl locks : 00000100  High depth limit  : 000
PropCtl. . : C                                   Low depth event . : N
Triggering                                       Low depth limit . : 000
Enabled  . : N        Transaction id.:
Type . . . :          Program id . . :
Max. starts: 0000     Terminal id  . :
Restart  . : N        Channel name . :
User data  :
           :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
            PF9=List  PF10=Queue
```

Figure 35. Model queue extended definition

The attributes on this screen are the same as those for local queues, and are described in "Local queue extended definition screen" on page 100, with the exception of the following attribute:

Def. type
Indicates whether a temporary dynamic or a permanent dynamic queue is created when this model queue is used to create a dynamic queue.

Valid values are:
T Temporary dynamic
P Permanent dynamic

When a dynamic queue is created (when an application uses the MQOPEN call to open a model queue), the dynamic queue inherits all the attributes of the model queue.

Note: Like local queues, the model queue extended definition specifies a CICS file name. The model queue, itself, does not use this file to store messages. Dynamic queues created using the model queue as a template use the file to store messages. Since a single model queue might be used to create numerous dynamic queues, each of these dynamic queues will use the same file to store its messages.

Create remote queue

Selecting "R" on the screen in Figure 31 on page 97 displays the screen shown in Figure 36 on page 107.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:00:38                Queue Definition  Record                     CIC1
MQWMQUE           QM - VSE.TS.QM1                                    A002

                      Remote Queue Definition

Object Name. . . . . . . . : VSE1.NT1.RQ1
Description line 1 . . . . : Remote queue to LQ1 on NT1
Description line 2 . . . . :

Put Enabled  . . . . . . . : Y   Y=Yes, N=No
Get Enabled  . . . . . . . : Y   Y=Yes, N=No

Remote Queue Name. . . . . : NT1.LQ1
Remote Queue Manager Name. : NT1.QM1
Transmission Queue Name. . : VSE1.XQ1







Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read   PF5=Add  PF6=Update
                                       PF9=List PF12=Delete
```

Figure 36. Remote queue definition

On this screen, the data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Put Enabled
A toggle that enables or disables MQPUT operations against this queue.

Get Enabled
A toggle that enables or disables MQGET operations against this queue. For remote queue definitions, the Get Enabled queue parameter is ignored because it is not possible to issue an MQGET call against a remote queue.

Remote Queue Name
The queue name on the remote WebSphere MQ system to which the definition in progress will refer.

Remote Queue Manager Name
The name of the remote WebSphere MQ system queue manager on which the remote queue is defined as a local queue. This name must be defined as a local transmission queue unless the Transmission Queue Name field is used.

Transmission Queue Name
The name of the local transmission queue to be used by WebSphere MQ to convey messages to this remote queue. If the field is left blank, the remote queue manager name is required to map to a local transmission queue.

Note: Some other operating systems, with which you could be communicating, may be case sensitive. You should read the information in "Uppercase translation" on page 27 before devising a name for a queue, channel or queue manager.

Navigation through the screens is dependent upon the PF keys.

Create alias queue

Selecting "AQ" on the screen in Figure 31 on page 97 displays the screen shown in Figure 37.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:09:40                Queue Definition  Record                     CIC1
MQWMQUE           QM - VSE.TS.QM1                                    A002

                       Alias Queue Definition

Object Name. . . . . . . . : EMPLOYEE
Description line 1 . . . . : Alias for EMPLOYEE.DETAILS queue
Description line 2 . . . . : with PUT inhibited.

Put Enabled  . . . . . . . : N   Y=Yes, N=No
Get Enabled  . . . . . . . : Y   Y=Yes, N=No

ALIAS Queue Name . . . . . : EMPLOYEE.DETAILS









Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

Figure 37. Alias queue definition

On this screen, the data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Put Enabled
A toggle that enables or disables MQPUT operations against this queue.

Get Enabled
A toggle that enables or disables MQGET operations against this queue.

Alias Queue Name
The name of another object already defined in the local configuration. This must be a local queue name. It cannot identify another alias.

Navigation through the screens is dependent upon the PF keys.

Create alias queue manager

Selecting "AM" on the screen in Figure 31 on page 97 displays the screen shown in Figure 38 on page 109.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:13:10                Queue Definition  Record                     CIC1
MQWMQUE           QM - VSE.TS.QM1                                    A002

                      Alias Queue Manager Definition

Object Name. . . . . . . . . : VSE.LOCAL1
Description line 1 . . . . . : Alias Queue Manager
Description line 2 . . . . . : for MQV1

Alias Queue Manager Name . . : MQV1
Transmission Queue Name. . . :









Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

Figure 38. Alias queue manager definition

On this screen, the definitions cannot be used in an MQCONN call; they may used for MQOPEN substitution only. The data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Alias Queue Manager Name
The name of a known queue manager. This can be a local transmission queue name, a remote queue manager name, or the local queue manager name. It cannot identify another alias.

Transmission Queue Name
The name of the local transmission queue to be used by WebSphere MQ to convey messages to this remote queue manager. If this field is left blank, the Alias Queue Manager Name field is required to map to a local transmission queue or to the local queue manager name.

Navigation through the screens is dependent upon the PF keys.

Creating alias reply queues

Selecting “AR” on the screen in Figure 31 on page 97 displays the screen shown in Figure 39 on page 110.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:20:24                Queue Definition  Record                     CIC1
MQWMQUE           QM - VSE.TS.QM1                                    A002

                      Alias Reply Queue Definition

Object Name. . . . . . . . : REPLYQ
Description line 1 . . . . : Alias reply definition
Description line 2 . . . . :

Alias Queue Name . . . . . : ANYQ
Alias Queue Manager Name . : VSE.TS.QM1









Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

Figure 39. Alias queue reply definition

On this screen, the definitions cannot be used in the MQOPEN call; they may only be used for reply queue name substitution with a MQPUT call. The data entry fields are:

Object Name
Filled in from the previous screen.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Alias Queue Name
The name of another object already defined in the local configuration. This can be a local queue name or a remote queue name. It cannot identify another alias.

Alias Queue Manager Name
The name of a known queue manager. This can be a local transmission queue name, a remote queue manager name, or the local queue manager name. It cannot identify another alias.

Navigation through the screens is dependent upon the PF keys.

Modifying and deleting queue definitions

Selecting 2 on the Configuration menu also allows you to modify or delete queue definitions.

Note: You use the same primary screens for modifying, deleting, or adding a queue.

Selecting an existing queue definition: To modify or delete an existing queue definition, you must select the definition on which to work, and display it.

To do this, select option 2 on the "Configuration Main Menu" screen to display the "Queue Main Options" screen (see Figure 31 on page 97) and use either the PF4, or PF9, function key.

PF4 is the Read key, and you use it to bring a specific queue definition to the screen as follows:

1. Enter the name of the desired queue in the Object Name field.
2. Press PF4 or Enter.
3. WebSphere MQ reads and displays the selected queue definition.

PF9 is the **LIST** key, and you use it to bring a specific queue definition to the screen as follows:

1. Press PF9.
2. The WebSphere MQ System displays a list of all defined queues (see Figure 40).
3. Select the desired queue by typing an "S" (or "X") next to its name, or by placing the cursor on the appropriate object.
4. Press PF4 or Enter.
5. WebSphere MQ reads and displays the selected queue definition.

```
07/06/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          MQBDTS
16:24:52                   Object List Screen                        CIC1
MQMMQUE                                                              A001


S Object                                              Type
  ANYQ                                                Local Queue
  EMPLOYEE                                            Alias Queue
  REPLYQ                                              Alias Reply
  REQUEST.MODEL                                       Model Queue
  SYSTEM.LOG                                          Local Queue
  VSE.LOCAL1                                          Alias Manager
  VSE1.NT1.RQ1                                        Remote Queue









Records found                 - Select one object name.

PF2 = Options   PF3 = Quit  PF4/Enter = Read
                PF7 = Backward  PF8 = Forward
```

Figure 40. Object list screen

Modifying an existing queue definition: When you have displayed the required queue definition, as described in "Selecting an existing queue definition" on page 110, you can modify any field in the definition. This may involve multiple screens to include all fields of the queue definition – see "Queue definitions" on page 96.

When you have made the changes you need, update the screen using the PF6 (UPDATE) function key.

Deleting an existing queue definition: When you have displayed the required queue definition, as described in "Selecting an existing queue definition" on page 110, you can delete it using the PF12 (DELETE) function key.

You will be asked to confirm that you want to delete the queue definition. You must press the PF12 function key again to delete the queue. If there are messages on the queue, it may take several seconds to delete these records.

Warning: Deleting a queue definition also deletes the queue's message data. If the message data is important, a queue should be emptied by normal application processing before it is deleted.

Channel definitions

Selecting 3 on the Configuration menu allows you to maintain (add, modify, or delete) channel definitions for the local installation of the WebSphere MQ System.

Note: The same screens are used to accomplish the three functions of adding, modifying, or deleting a channel definition; the required action being selected through the function keys. The following sections present the screens that you see if you are adding a new channel definition.

"Modifying and deleting channel definitions" on page 115 explains how you modify or delete a queue.

The screen shown in Figure 41 is displayed to create a channel definition:

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:01:29                    Channel Record           DISPLAY         CIC1
MQWMCHN                                                              A001
Channel  : MY.SDR
 Desc. . :
 Protocol: T (L/T)  Type : S (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000000
 Get retry number . . . . . : 00000000   Short retry interval  . . : 000000000
 Get retry delay (secs) . . : 00000000   Long retry interval . . . : 000000000
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000000000
 Property control . . . . . : C
 Transmission queue name. . : MY.XMITQ
 TP name. . :
Sender/Receiver/Server/Requester
 Connection : 1.1.1.1(1414)
 Max Messages per Batch . . : 000010     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 065535     Split Msg(Y/N)  . . . . . : Y
 Max TCP/IP Wait  . . . . . : 000300     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

Figure 41. Channel record

On this screen, the data entry fields are:

General

Channel name
The name of the channel to be defined.

Description
The description of the channel. This field is for documentation purposes only, and does not affect channel operation.

Protocol
The protocol being used by the selected channel, which can be LU 6.2, or TCP/IP.

Type
S: Sender channel.
R: Receiver channel.
V: Server channel.
Q: Requester channel.
C: Server-connection (SVRCONN) channel.

Enable
Enable the channel for communications at initialization time.

Sender and Server channels

The following parameters are relevant to all sender and server channels.

Remote TCP/IP Port
The port number that WebSphere MQ uses for accepting TCP/IP connection requests. For the following type of channels the port number is

SENDER
Port number of the remote listener

SERVER
Port number of the remote listener

REQUESTER
Port number of the remote listener

RECEIVER
Not required

SVRCONN
Not required

The number reserved for WebSphere MQ is 1414, although any unreserved number can be used.

Get retry number
The number of Get retries when queue is empty.

Get retry delay
The time between retries (in seconds).

Convert Msgs
A field that identifies whether message data is converted before it is sent to a remote queue manager. To convert message data, set this field to Y. Data is converted to the code page of the remote host only if the code page is supported.

Property control
Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender and Server. Permitted values are:

C (COMPAT) This is the default value.

N (NONE) All properties of the message, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

A (ALL) All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

Transmission Queue Name
The name of the transmission queue. Required for the sender, optional for the receiver.

TP Name
A 64-character field identifying the remote task ID of the receiver on the remote CICS region, or a TPNAME on the remote system (for example, AMQCRS6A). This is required by the sender for LU 6.2 use only. This field is not recognized for TCP/IP.

Note: Although the TPNAME can be up to 64 bytes in other WebSphere MQ products, on WebSphere MQ for z/VSE it can have a maximum of 4 bytes only.

Short/Long retry count
Number of attempts to connect when initial connection is unsuccessful.

Short retry interval
Time in seconds between connection retry attempts. When an initial connection attempt is unsuccessful, the queue manager will retry Short/Long retry count times, with a Short retry interval attempts between each retry.

Long retry interval
Time in seconds between connection retry attempts after the short retry interval attempts have been exhausted. Once the long retry attempts have been exhausted, the connection is deemed failed.

Batch interval
For sender and server channels, the batch interval (in milliseconds) is used to ensure a batch of message is not deemed complete until the interval has expired.

Normally, messages are transmitted in batches of a size specified by the channel's Batch Size attribute. When the batch is complete, the queue manager negotiates with the remote system to commit the batch.

In the event that messages arrive slowly on a transmission queue, it is possible for the queue manager to deem a batch complete when the transmission queue is empty, even though messages are still arriving. The end of batch channel processing imposes some processing overhead.

Consequently, the batch interval attribute can be used to make the queue manager wait a period of time before initiating end of batch processing.

Sender, Server, Receiver and Requester channels

The following parameters are relevant to all channels except server-connection channels.

Connection
The name of the LU 6.2 connection, or for TCP/IP sender channels, the remote host name or IP address.

For TCP/IP sender channels, the remote port number can be appended (in parentheses) to the connection name. For example:

```
my.remote.host(1414)
```

If the port number is not appended to the connection name, the queue manager will use the value specified by the Remote TCP/IP port parameter.

Max Messages per Batch
The mutually accepted maximum number of messages per batch to be transmitted.

Message Sequence Wrap
The mutually agreed maximum messages count before the count sequence starts over.

Dead Letter Store
A field that identifies whether inbound messages are written to the dead letter queue whenever an inbound message cannot be written to its intended target queue. The dead letter queue is identified in your Global System Definition.

Max Transmission Size
The mutually accepted maximum number of bytes per transmission.

The maximum transmission size for LU6.2 channels is 32000. For TCP/IP the maximum is 65535.

Max Message Size
The mutually accepted maximum number of bytes per message. The Maximum Message Size must be bigger than the application message size, plus approximately 1,000 bytes for the WebSphere MQ header.

Split Msg
This **must** be set to `Y` if messages longer than the maximum transmission size for the channel are to be sent across the channel.

Max TCP/IP Wait
The maximum number of second that a Message Channel Agent (MCA) should wait to receive TCP/IP data before terminating the connection with an error. See “Bullet-proof channels” on page 74 for more information.

Channel statistics
Indicates whether or not channel statistics should be collected for the channel.

Channel monitoring
Indicates whether or not channel monitoring information should be collected for the channel.

Modifying and deleting channel definitions

Selecting 3 on the Configuration menu also allows you to modify or delete channel definitions.

Note: You use the same primary screens for modifying, deleting, or adding a channel.

Selecting an existing channel definition

To modify or delete an existing channel definition, you must select the definition on which to work, and display it.

To do this, select option 3 on the "Configuration Main Menu" screen to display the "Channel Record" screen (see Figure 41 on page 112), and use either the PF4 or PF9 function key.

PF4 is the Read key, and you use it to bring a specific channel definition to the screen as follows:

1. Enter the name of the desired channel in the Channel Name field.
2. Press PF4 or Enter.
3. WebSphere MQ reads and displays the selected channel definition.

PF9 is the List key, and you use it to bring a specific channel definition to the screen as follows:

1. Press PF9.
2. WebSphere MQ displays a list of all defined channels (see Figure 42).
3. Select the desired channel by typing an "S" next to the channel name.
4. Press PF4 or Enter.
5. WebSphere MQ reads and displays the selected channel definition.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
11:10:12                   Channel List                              CIC1
MQWMCHN                                                              A002
S CHANNEL NAME         TYPE STATUS   LAST MSN   CHECKPOINT
  MQTS.TO.MQ23           S  ENABLE           2  2002/12/23 10:49:41  TCP
  MQ23.TO.MQTS           R  ENABLE           1  2002/12/23 10:49:41  TCP
  MVSA_TO_VSEA           R  ENABLE           0  2002/12/23 10:49:41




ENTER 'S' to select Channel
F2=Return  PF3=Quit  PF4=Read  F7=Backward  PF8=Forward
```

Figure 42. Channel list

On this screen, the display fields are:

Channel Name
The names of all channels.

Type Type is sender (S), receiver (R), server (V), requester (Q), or svrconn (C).

Status Channel may be enabled (ENABLE) or disabled (DISABL).

Last MSN
The last checkpointed message sequence number of the channel.

Checkpoint
The time of the last checkpoint.

Modifying an existing channel definition

When you have displayed the required channel definition, as described in "Selecting an existing channel definition" on page 115, you can modify any field in the definition.

When you have made the changes you need, update the screen using the PF6 (Update) function key.

Deleting an existing channel definition

When you have displayed the required channel definition, as described in "Selecting an existing channel definition" on page 115, you can delete it using the PF12 (Delete) function key.

You will be asked to confirm that you want to delete the channel definition. You must press the PF12 function key again to delete the channel.

Setting channel SSL parameters

If a channel requires secure sockets layer (SSL) services, then press PF10 (SSL) to display the Channel SSL Parameters screen:

```
11/22/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:34:00                  Channel SSL Parameters                     CIC1
MQMMCHN                                                              A001

     Channel Name: VSE1.TCP.AIX2          Type: S

     SSL Cipher Specification. : 0A       (2 character code)
     SSL Client Authentication : O        (Required or Optional)

     SSL Peer Attributes:
     > CN=IBM*                                                        <
     >                                                                <
     >                                                                <
     >                                                                <









SSL channel parameters displayed.
    PF2 = Return   PF3 = Quit     PF4 = Read      PF6 = Update
```

Figure 43. Channel SSL parameters

On this screen, the data entry fields are:

SSL Cipher Specification
The SSL Cipher Specification identifies a 2-character code for any of the supported SSL Version 3.0 ciphers. At the time of publication, these include:
01 for NULL MD5
02 for NULL SHA
08 for DES40 SHA for Export
09 for DES SHA for US

0A for Triple DES SHA for US
62 for RSA_EXPORT1024_DESCBC_SHA
2F for RSA_AES128CBC_SHA
35 for RSA_AES256CBC_SHA

It should be noted that the bit-size of an RSA private key can affect the list of valid cipher specifications. For more information, refer to relevant SSL product documentation.

It is expected that the list of supported ciphers will expand over time. Consequently, this field is not validated. Any non-blank value is accepted as a valid configuration value. However, if an unsupported code is used, the channel will fail on any attempted use. For this reason, SSL services documentation relevant to your current environment should be checked prior to setting this field.

This field, and this field alone, determines whether or not the channel will apply SSL services. In other words, the other SSL parameters in the channel configuration are ignored if this field is not set (that is, if it is left blank). Consequently, this field can be used to activate and deactivate SSL services on a channel without requiring the other parameters being reset.

SSL Client Authentication

SSL client authentication can be one of:

R Required
O Optional

If required, WebSphere MQ checks that the remote SSL partner (WebSphere MQ message channel agent or MQ client) provided a X.509 PKI certificate during SSL initial negotiation. If the remote partner fails to send a certificate, and one is required, the channel is terminated. If client authentication is optional, WebSphere MQ ignores whether or not a certificate was received.

SSL Peer Attributes

SSL peer attributes is a 256-character case-sensitive field that can be used to ensure a remote partner's certificate contains identifiable attributes. This requires that the remote partner provided a certificate during SSL initial negotiation. If the remote partner fails to provide a certificate, then any check against the SSL Peer Attributes field will fail, and the channel will be terminated. The SSL Peer Attributes field expects a value (if any) in the form:

```
key=value,key=value, etc.
```

where *key* is one of the supported keywords (see below), the equal sign (=) is mandatory, and *value* is a value relative to the keyword that is expected to match the remote partner's certificate. Supported keywords include:

CN Common name
C Country
ST State or province
L Locality
O Organization
OU Organization Unit
SERIAL
Serial number

For example:

```
C=US,O=IBM
```

In this example, the Country field of the remote partner's X.509 PKI certificate must be “US”, and the Organization field must be “IBM”. Note that values can include imbedded blanks and wild cards (*). Values with embedded blanks must be enclosed in double quotes ("). If used, only one wild card can be used in each value, and only at the end of that value. For example:

```
C=US,O="IBM GSA *",CN=www.ibm.*
```

Following SSL negotiation, WebSphere MQ checks that the remote partner's X.509 PKI certificate matches the SSL peer attributes specified by this field. If they match, channel activity proceeds with SSL services enabled. If they do not match, the channel is terminated.

Setting channel exit parameters

If the channel requires exit processing during its operation, Then press PF11 (Ext) to display the Channel Exit Settings screen:

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:54:10                  Channel Exit Settings                     CIC1
MQWMCHN                                                             A000

Channel name . . . : VSE9.IP.SOL7
Channel type . . . : Sender

Security exit name :
Security exit data :









Channel exit settings displayed.
F2=Return  PF3=Quit  PF4=Read  F6=Update    PF10=SndX PF11=RcvX PF12=MsgX
```

Figure 44. Channel Exit settings

On this screen, the Channel name and type are display-only fields. The data entry fields are:

Security exit name
: The name of the security exit program that is invoked during the initial negotiation of the channel.

Security exit data
: Optional data that is passed to the security exit program on each invocation.

The Channel Exit Settings screen allows channel exit programs, and associated data, to be set for send, receive, security and message exits. Send, Receive and Message exits are configurable using function keys PF10, PF11 and PF12 respectively.

Configuration for send, receive and message exits is similar. For example, using PF10 to configure send exits presents the following screen:

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:59:33               Channel Send Exit Settings                   CIC1
MQWMCHN                                                             A000

Channel name . . . : VSE9.IP.ZOSX
Channel type . . . : Sender

  Exitname           Exit Data
> SNDEXIT1 <       > Send exit data 1                <
> SNDEXIT2 <       > Send exit data 2                <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <
>          <       >                                 <

Channel exit settings displayed.
F2=Return  PF3=Quit  PF4=Read  F6=Update    PF10=SndX PF11=SecX PF12=MsgX
```

Figure 45. Channel chained-exit settings

Note that you can configure up to 8 send exits and associate exit data. These exits, if configured, are called in the sequence listed. The same is true for receive and message exits.

Channel exit names can be 1-8 characters, and follow the naming standard for any program defined in the CICS CSD.

For more information on channel exits, see “Channel exits” on page 55.

Code page definitions

Selecting 4 on the Configuration menu allows you to define global parameters and maintain a set of user-defined code pages for data conversion.

Figure 46 on page 121 shows the screen that is displayed when you select 4 from the Configuration menu.

```
07/07/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          MQBDTS
09:58:32                  Code Page Definition                       CIC1
MQMMCPG                Data Conversion Information                   A001



          Default ASCII code page....: 0850



          Default EBCDIC code page...: 1047



          Convert EBCDIC newline.....: L    L=Ascii NL, T=Table, I=ISO




Record initialized - New record.
 PF2 = Return         PF3 = Quit     PF4/ENTER = Read    PF5 = Add PF6 =Update
                                     PF9 = List CodePages
```

Figure 46. Data conversion definitions

To change values on this screen, use PF6 (Update). To add user-defined code pages, use PF5 (Add).

On this screen, the data entry fields are:

Default ASCII code page
Code page used for default conversion when a code page that represents an ASCII code page fails normal conversion.

The default data conversion process is used if data conversion fails and the default ASCII code page is valid. Default conversion takes place only if the conversion is from ASCII to EBCDIC, or EBCDIC to ASCII. Otherwise, data is passed without conversion.

A value of 0 means no default conversion.

Default EBCDIC code page
Code page used for default conversion when a code page that represents an EBCDIC code page fails normal conversion.

The default data conversion process is used if data conversion fails and the default EBCDIC code page is valid. Default conversion takes place only if the conversion is from ASCII to EBCDIC, or EBCDIC to ASCII. Otherwise, data is passed without conversion.

A value of 0 means no default conversion.

Convert EBCDIC newline
This value is used only when converting EBCDIC to ASCII. Valid values are:

L: EBCDIC NL is converted to ASCII LF. This is the default behavior and how all V5.0 WebSphere MQ products behave.

T: EBCDIC NL is converted to whatever value is specified in the supplied conversion table.

I: EBCDIC NL is converted to whatever value is specified inthe supplied conversion table. If the source is an ISO code page, the conversion is the same as L (NL to LF).

See the *WebSphere MQ Application Programming Reference* for more details regarding conversion of EBCDIC newline characters.

Create a user-defined code page

Selecting option 4 on the Configuration menu also allows you to create user-defined code pages.

To add a new user-defined code page, select option 4 on the configuration menu to display the Code Page Definition screen (see Figure 46 on page 121), then press PF5.

The screen shown in Figure 47 is displayed.

```
07/07/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          MQBDTS
10:42:50                 User Code Page Definition                   CIC1
MQMMCPG                                                              A001




 Code Page Number . . . . . : 1234
 Description Line 1 . . . . : Special ASCII code page
 Description Line 2 . . . . :

 Type . . . . . : S      S=SBCS, D=DBCS, M=MIXED
 Encoding . . . . . : A      E=EBCDIC, A=ASCII, I=ISO,
                            U=UCS2, T=UTF, C=EUC



Record being added   - Press ADD key again.

PF2=Data Conv   PF3 = Quit PF4/ENTER = Read   PF5 = Add          PF6 = Update
                PF9 = List                                       PF12= Delete
```

Figure 47. User code page definition

On this screen, the data entry fields are:

Code Page Number
Four digit number that uniquely identifies the user-defined code page. You cannot redefine a system-defined code page that already exists in LE/VSE.

Description lines 1 & 2
Text fields for operator use only. They may each be up to 32 characters.

Type S: Single Byte Character Set (SBCS)

D: Double Byte Character Set (DBCS)

M: Mixed SBCS/DBCS

Encoding
E: EBCDIC

A: ASCII

I: ISO

U: UCS-2 (Unicode)

T: UTF-8 (Unicode)

C: EUC

For more information on these types and encodings, see the IBM manual *Character Data Representation Architecture*, SC09-1390.

For a user code page to work, the underlying LE/VSE code converter must exist as a CICS phase. WebSphere MQ for z/VSE uses LE/VSE for application message data conversion. See Chapter 7, "Message data conversion," on page 217 for more details on message data conversion.

Modifying and deleting user-defined code pages

Selecting option 4 on the Configuration menu also allows you to modify or delete user code page definitions.

Note: You use the same primary screens for both modifying and deleting a user code page definition.

Additionally, you can use the PF9 function key (List) from either the Code Page Definition screen, or the User Code Page Definition screen.

PF9 displays the code page Object List screen shown in Figure 48.

```
07/07/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          MQBTDS
04:14:06                   Object List Screen                        CIC1
MQMMCPG                                                              A001


S Object                                          TYPE
  1123                                            SBCS Codepage
  1208                                            Mixed Codepage
  1308                                            SBCS Codepage
  3254                                            SBCS Codepage
  3722                                            SBCS Codepage
  4355                                            Mixed Codepage
  4545                                            SBCS Codepage
  4567                                            SBCS Codepage
  5657                                            SBCS Codepage
  6775                                            SBCS Codepage
...More



Records Found - Select one Object Name

PF2 =Data Conv  PF3 = Quit  PF4/Enter = Read
                PF7 = Backward  PF8 = Forward
```

Figure 48. Code page object list screen

Modifying an existing code page definition

From the displayed list, type an S next to the code page number you want to modify, or position the cursor on the entry, then press Enter.

When the required code page definition is displayed, you can edit modifiable fields and update the entry using PF6.

Deleting an existing code page definition.

From the displayed list, type an S next to the code page number you want to delete, or position the cursor on the entry, then press Enter.

When the required code page definition is displayed, use PF12 to delete the entry.

You are asked to confirm that you want to delete the definition. You must press the PF12 key again to delete the code page definition.

Listener definitions

Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels. Listener processes can be configured using the master terminal transaction (MQMT), Programmable Command Format (PCF), or MQSeries Command (MQSC) requests.

Listener objects are WebSphere MQ objects that allow you to manage the starting and stopping of listener processes from within the scope of a queue manager. By defining attributes of a listener object, you:

1. Configure the listener process.
2. Specify whether the listener process automatically starts and stops when the queue manager starts and stops. In WebSphere MQ for z/VSE, all running listeners are stopped when the queue is stopped regardless of the control mode.

Listener objects can be created, modified and deleted using the master terminal transaction (MQMT), Programmable Command Format (PCF), or MQSeries Command (MQSC) requests.

When WebSphere MQ for z/VSE is initialized, if no listener objects are found, a listener "LISTENER.TCP" is created using the old listener port number that was used prior to the introduction of listener object support. The "TCP/IP listener port" field in the communications settings panel below has been removed.

Using the Master terminal transaction (MQMT):

1. Select **1. Configuration** from the Master Terminal Main Menu.
2. Select **1. Global System Definition** from the Configuration Main Menu.
3. On the Global System Definition panel, press PF9 (Communications) to display the Communications Settings panel.
4. Press PF10 (Listeners) to display the panel shown here:

```
12/11/2010        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:42:13                    Listener Record          UPDATE         CIC1
MQWMSYS                                                             A002


Listener name . . . . :
Description . . . . . :
                      :

Control . . . . . . . :        (M=Manual, Q=Qmgr start/stop)
Transmission protocol :        (T=TCP)
Port  . . . . . . . . :
IP address. . . . . . :
Backlog . . . . . . . :

Listener status . . . :
Alteration date . . . :
Alteration time . . . :



Enter object name.
PF2=Return PF3=Quit PF4=Read PF5=Add    PF6=Update   PF9=List
                             PF10=Start PF11=Stop    PF12=Delete
```

On the Listener Record panel, the fields are:

Listener name
The name of the listener object. The name may be up to 48 characters, it must be unique, and it must conform to WMQ object naming requirements.

Description
Optional description of the purpose of the listener. The description can be up to 64 characters.

Control
Determines whether the listener is started:

M Manually.

Q When the queue manager starts, and stopped when the queue manager is stopped.

S When the queue manager starts, but not stopped when the queue manager is stopped

Transmission protocol
Transmission protocol used by the listener. WebSphere MQ for z/VSE only supports:

T TCP/IP

Port Port number on which the listener listens for connections.

IP address
The name of the computer on which the listener listens for connections. You can use either of these formats:

- IPv4 dotted decimal.
- Fully-qualified hostname.

If no value is specified, the listener listens on all available IPv4 addresses.

Backlog
The maximum number of concurrent connection requests that the listener supports. The default value is 10.

Listener status
Read-only. This attribute shows the current status of the listener. It can be RUNNING or STOPPED.

Alteration date
Read-only. This attribute shows the date on which the listener's attributes were last altered.

Alteration time
Read-only. This attribute shows the time at which the listener's attributes were last altered.

Function keys
The listener screen supports these functions keys:

PF2 Return to previous screen.
PF3 Quit master terminal transactions.
PF4 Display the listener object record.
PF5 Add a new listener object.
PF6 Update the current listener object.
PF9 List all defined listener objects.
PF10 Start the listener object.
PF11 Stop the listener object.
PF12 Delete the current listener object.

Press PF9 to list listeners.

```
2010/12/11        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:44:56                    Listener List                           CIC1
MQWMSYS                                                             A002
S LISTENER NAME                                      CTL  XMIT  PORT  STATUS
  LISTENER.TCP                                       Q    T     1439  RUNNING
```

```
Select object or page up/down list
F2=Return  PF3=Quit  PF4=Read  F7=Backward  PF8=Forward  PF9=List
```

On the Listener List panel, the fields are read-only, except for "S" which allows you to select a particular object for processing:

S Press Enter, or enter any character next to an object to select it for processing.

LISTENER NAME
The name of the listener object.

CTL Indicates whether the listener is started:
- **M** Manually.
- **Q** When the queue manager starts, and stopped when the queue manager is stopped.
- **S** When the queue manager starts, but not stopped when the queue manager is stopped.

XMIT Transmission protocol used by the listener. WebSphere MQ for z/VSE only supports TCP.

PORT Port number on which the listener listens for connections.

STATUS
Indicates whether the listener is STOPPED or RUNNING.

Selecting a listener object displays the object's details. For example:

```
2010/12/11        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:50:52                    Listener Record          DISPLAY         CIC1
MQWMSYS                                                              A002


Listener name . . . . :  LISTENER.TCP
Description . . . . . :  AUTO CREATED BY QUEUE MANAGER...
                      :

Control . . . . . . . :  Q     (M=Manual, Q=Qmgr start/stop)
Transmission protocol :  T     (T=TCP)
Port  . . . . . . . . :  01439
IP address. . . . . . :
Backlog . . . . . . . :  0010

Listener status . . . :  RUNNING
Alteration date . . . :  2010-05-26
Alteration time . . . :  10:38:24



Requested record displayed.
PF2=Return PF3=Quit PF4=Read PF5=Add    PF6=Update   PF9=List
                             PF10=Start PF11=Stop    PF12=Delete
```

Service definitions

Service objects are a way of defining programs to be executed when a queue manager starts or stops. The programs can be split into these types:

Servers

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. Only one instance of a server process can be executed concurrently. While running, the status of a server process can be monitored using the MQSC command, DISPLAY SVSTATUS. The status is RUNNING until a STOP SERVICE is issued. WebSphere MQ for z/VSE does not attempt to check on the status of the server task.

Typically, server service objects are definitions of programs such as dead letter handlers or trigger monitors. However the programs that can be run are not limited to those supplied with WebSphere MQ. Additionally, a server service object can be defined to include a command that is run when the specified queue manager is shutdown to end the program.

Commands

A command is a service object that has the parameter SERVTYPE specified as COMMAND. A command service object is the definition of a program that is executed when a specified queue manager is started or stopped. Multiple instances of a command process can be executed concurrently. Command service objects differ from server service objects in that once the program is executed the queue manager does not monitor the program. Typically, command service objects are definitions of programs that are short-lived and perform a specific task such as starting one or more other tasks.

Service objects can be created, modified, and deleted using the master terminal transaction (MQMT), Programmable Command Format (PCF), or MQSeries Command (MQSC) requests.

Using the Master terminal transaction (MQMT):

1. Select **1. Configuration** from the Master Terminal Main Menu.
2. Select **1. Global System Definition** from the Configuration Main Menu.
3. On the Global System Definition panel, press PF9 (Communications) to display the Communications Settings panel.
4. Press PF11 (Services) to display the panel shown here:

```
2010/12/11        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
08:03:43                    Service Record           UPDATE          CIC1
MQWMSYS                                                              A002

Service name. . . . . :
Description . . . . . :
                      :

Service type  . . . . :        (S=Server, C=Command)
Control . . . . . . . :        (M=Manual, Q=QMgr start/stop, S=QMgr start)
Start command . . . . :
Start args  . . . . . :
                      :
Stop command  . . . . :
Stop args   . . . . . :
                      :

Service status. . . . :
Alteration date . . . :
Alteration time . . . :

Enter object name.
PF2=Return PF3=Quit PF4=Read PF5=Add    PF6=Update   PF9=List
                             PF10=Start PF11=Stop    PF12=Delete
```

On the Services screen, the fields are:

Service name
The name of the service object. The name can be up to 32 characters, it must be unique and it must conform to WMQ object naming requirements.

Description
Optional description of the purpose of the service. The description can be up to 64 characters.

Service type
Indicates the type of service:
S Server. To enable only one instance of the service to run at a time.
C Command. To enable multiple instances of the service to run at a time.

Control
Determines whether the service is started:
M Manually.
Q When the queue manager starts, and stopped when the queue manager is stopped
S When the queue manager starts, but not stopped when the queue manager is stopped

Start command
The name of a CICS transaction.

Start args
Data area to be passed to the CICS transaction by means of COMMAREA when started.

Stop command
The name of a CICS transaction.

Stop args
Data area to be passed to the CICS transaction by means of COMMAREA when stopped.

Service status
Read-only. This attribute shows the current status of the service. It can be RUNNING or STOPPED. This only applies to SERVER type service.

Alteration date
Read-only. This attribute shows the date on which the service's attributes were last altered.

Alteration time
Read-only. This attribute shows the time at which the service's attributes were last altered.

Function keys
The Services screen supports these functions keys:
PF2 Return to previous screen.
PF3 Quit master terminal transactions.
PF4 Read and display the service object record.
PF5 Add a new service object.
PF6 Update the current service object.
PF9 List all defined service objects.
PF10 Start the current service object.
PF11 Stop the current service object.
PF12 Delete the current service object.

List all service objects

From the Service screen, PF9 lists all defined service objects:

```
2010/12/11        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:06:10                    Services List                           CIC1
MQWMSYS                                                             A002
S SERVICE NAME                                      CONTROL   TYPE     STATUS
  BATCH.INTERFACE                                   QMGR      SERVER   RUNNING
  BATCH.INTERFACE.START                             MANUAL    COMMAND
  BATCH.INTERFACE.STOP                              MANUAL    COMMAND
```

```
Select object or page up/down list
F2=Return  PF3=Quit  PF4=Read  F7=Backward  PF8=Forward  PF9=List
```

On the Services List panel, the fields are read-only except for "S", which allows you to select a particular object for processing:

S Press Enter or enter any character next to an object to select it for processing.

SERVICE NAME
The name of the service object.

CONTROL
Indicates whether the service is started:

- **M** Manually.
- **Q** When the queue manager starts, and stopped when the queue manager is stopped.
- **S** When the queue manager starts, but not stopped when the queue manager is stopped.

TYPE Service type. Value can be:

- **S** Server. Enabled for only one instance of the service to run at a time.
- **C** Command. Enabled for multiple instances of the service to run at a time.

STATUS
Indicates whether the service is STOPPED or RUNNING only for service type of SERVER.

Start service object

From the Services panel, PF10 allows you to start the selected service. The behaviour of PF10 is dependant on the service type. For a service type of Server, if the service is not running, the service is started. If the service is running, the start request is ignored. For a service type of Command, PF10 starts the service.

Stop service object

From the Services screen, PF11 allows you to stop the selected service. The behavior of PF11 is dependant on the service type. For a service type of Server, if the service is running, the service is stopped. If the service is not running, the stop request is ignored. For a service type of Command, PF11 invokes the Command using the Stop args.

Namelist definitions

Selecting 5 on the Configuration menu allows you to define, modify and delete namelists.

Figure 49 shows the screen that is displayed when you select 5 from the Configuration menu.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:06:59                   Namelist Record           UPDATE          CIC1
MQWMNAM                                                              A001

   Object name:
   Description:

   Name:                                          (1..12)
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   ...more

Enter Namelist Name.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF7=Back PF8=Fwd PF9=List PF12=Del
```

Figure 49. Namelists Definitions

On this screen, the data entry fields are:

Object name
The name of the namelist object. Namelist names can be 1 to 48 characters, and must conform to the MQ object naming standards described in section "Object names" on page 3.

Description
Namelist description. This field is optional, and allows up to a 64-character description.

Names
Namelist names. WebSphere MQ for z/VSE allows a namelist to specify up to 256 names. Blank names are ignored. The PF7 and PF8 function keys allows you to page back and forward, respectively, through the names.

Create a namelist

Selecting option 5 on the Configuration menu also allows you to create namelists.

To add a new namelist, select option 5 on the Configuration menu to display the Namelist Definition screen. Enter the new namelist object name, an optional description, and a list of names, then press PF5 (ADD).

The screen shown in Figure 50is displayed.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:19:10                    Namelist Record          ADD             CIC1
MQWMNAM                                                              A001

   Object name:  MY.NAMELIST
   Description:  My namelist description

   Name:                                          (1..12)
   > MY.NAME.1                                          <
   > MY.NAME.2                                          <
   > MY.NAME.3                                          <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   >                                                    <
   ...more

Namelist record added OK
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF7=Back PF8=Fwd PF9=List PF12=Del
```

Figure 50. Creating a namelist

Note that if your new namelist has more than 12 names, you must ADD the initial 1 to 12 names first, then modify the namelist to add any additional names.

Displaying existing namelists

From the Namelist definition screen you can use the PF9, List, function key to display a list of all existing namelist objects.

PF9 displays the following screen:

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:28:52                    Namelist List                            CIC1
MQWMNAM                                                              A001
S NAMELIST                                                    COUNT
  MY.NAMELIST                                                     3
  NAME.LIST.WITHOUT.NAMES                                         0
  SYSTEM.DEFAULT.NAMELIST                                         0
  TEST.NAME1                                                      4
  TEST.NAME2                                                      6
  TEST.NAME256                                                  256
ENTER 'S' to select Namelist
F2=Return  PF3=Quit  PF4=Read  PF7=Backward  PF8=Forward
```

Figure 51. Listing namelists

On this screen you can scroll through the namelist object names using the PF7 and PF8 function keys.

The left-hand column allows you to select a particular namelist name. Selecting a namelist name, and pressing Enter, returns to the Namelist definition screen with that namelist's details displayed.

Modifying and deleting namelists

You can use the PF9, List, function key to select a particular namelist object, or you can enter the name of the namelist in the object name field. Pressing Enter displays the attributes of the namelist specified in the object name field.

To modify a namelist, change the relevant fields and press PF6, UPDATE. You can page through the namelist names using PF7 and PF8. Erasing or blanking out a name results in that name being removed from the list.

To delete a namelist, use PF12, DELETE. This prompts for confirmation before deleting the namelist object. To confirm, press PF12 again.

Global system definition display

Selecting 6 on the main menu allows you to view the attributes defined for the local queue manager, and all system-wide parameters, through the screen shown in Figure 52 on page 134, which is display only:

```
2013/01/25        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
05:46:06                  Global System Definition                  CIC1
MQWMSYS                 Queue Manager Information                   A002
Queue Manager . . . . . . . : PTHVSEC
Description Line 1. . . . . : ZVSE 4.3 SYSTEM ON PTHVSEC
Description Line 2. . . . . : ...
                          System Values
Maximum Connection Handles.: 00000100       System Wait Interval : 00000030
Maximum Concurrent Queues .: 00000100       Max. Recovery Tasks  : 0000
Allow TDQ Write on Errors  : Y   CSMT       Local Code Page  . . : 01047
Allow Internal Dump  . . . : Y              Subsystem id . . . . : MQV1
                                            Channel Auth Enabled : Y
                       Queue Maximum Values
Maximum Q Depth . . . . . . : 00010000      Maximum Global Locks.: 00001000
Maximum Message Size. . . . : 00409600      Maximum Local Locks .: 00001000
Maximum Single Q Access . .: 00000100       Max Properties Length: 00004094
                       Global QUEUE /File Names
Configuration File. : MQFCNFG
LOG Queue Name. . . : SYSTEM.LOG
Dead Letter Name. . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name. : SYSTEM.MONITOR
Requested record displayed.
PF2=Return PF3=Quit PF4/ENTER=Refresh    PF6=Update
                    PF9=Communications  PF10=Log    PF11=Events  PF12=Exits
```

Figure 52. Global system definition display

To return to the configuration main menu, press the PF2 key. To show display-only screens for system-wide communications settings, log and trace settings, event settings or API exit settings, press PF9 (Com), PF10 (Log), PF11 (Evt), or PF12 (Ext) respectively.

Queue definition display

Selecting 7 on the main menu allows you to view existing queue definitions.

Note: This function allows you to see the queue definition, not the current queue status. To see the current queue information, see "Monitor queues" on page 145.

This operation is identical to the modify queue and delete queue operations (as described in "Modifying and deleting queue definitions" on page 110), except that the maintenance function keys PF5 (Add), PF6 (Update), and PF12 (Delete), are not available.

Channel definition display

Selecting 8 on the main menu allows you to view existing channel definitions.

Note: This function allows you to see the channel definition, not the current channel status. To see the current channel information, see "Monitor channel" on page 147.

This operation is identical to the modify channel and delete channel operations (as described in "Modifying and deleting channel definitions" on page 115), except that the maintenance function keys PF6 (Update) and PF12 (Delete), are not available.

Code page definition display

Selecting 9 on the Configuration menu allows you to view existing code page defaults and user-defined code page definitions.

This operation is identical to the modify and delete code page operations, as available using option 4 of the Configuration menu, except that the maintenance function keys PF6 (Update) and PF12 (Delete) are not available.

Operations functions

Selecting option 2 (Operations) from the master terminal main menu (see Figure 24 on page 80) displays the following screen:

```
12/27/2008       IBM WebSphere MQ for z/VSE Version 3.0.0            TSMQ300
13:02:40              *** Operations Main Menu ***                   CIC1
MQWMOPR                                                              A002
                             SYSTEM IS ACTIVE

                          1. Start / Stop Queue(s)

                          2. Open / Close Channel(s)

                          3. Reset Message Sequence Number

                          4. Initialization / Shutdown of System

                          5. Maintain Queue Message Records

                 Option:




    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Select   PF2=Return   PF3=Quit
```

Figure 53. Operations main menu

On this screen, selections correspond to available operator control functions.

Start/Stop queue

Selecting 1 on the operations menu allows you to start or stop processing for a queue.

This differs from setting the queue's Get Enabled or Put Enabled option to "No" in that the Start and Stop functions apply universally to all processes attempting to access a local queue.

The Get Enabled and Put Enabled functions can be selectively applied to aliases and remote queue definitions.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:07:54               Start / Stop Queue                            CIC1
MQWMSS                                                               A002

                           System Information
          System Status    : SYSTEM IS ACTIVE
          Queue Status     : Queuing System is active.
          Channel Status   : Channel System is active.
          Monitor Status   : Monitor is not active.
                          Single Queue Request
          Queue Name       :
          Function         :         S=Start, X=Stop,  R=Refresh from Config
          Mode             :         I=Inbound, O=Outbound, B=Both

          INBOUND Status   :
          OUTBOUND Status  :

                           Queuing System Request
          Function         :         S=Start, X=Stop, or M=Monitor


Please enter a Queue name.
Enter=Display  PF2=Return  PF3=Exit  PF6=Update
```

Figure 54. Start / Stop queue control screen

On the Start / Stop Queue screen shown in Figure 54, the fields are:

System Information

System Status
Reflects the status of the system. This is normally ACTIVE, unless the system has not been initialized, or has been shut down. When this occurs, the field reads SYSTEM IS SHUTDOWN.

Queue Status
Reflects the status of the queuing system. This is normally ACTIVE, unless the system has not yet been initialized or all the queues have been stopped. When this occurs, the field reads QUEUING SYSTEM IS STOPPED.

Channel Status
Reflects the status of the channels. This is normally ACTIVE, unless the system has not yet been initialized or all the channels have been closed. When this occurs, the field reads CHANNEL SYSTEM IS CLOSED.

Monitor Status
Reflects the status of the system monitor.

Single Queue Request

Queue Name
The name of a specific queue to start or stop

Function
The function to be performed, as follows:

S is to start:
- A stopped local queue.
- The associated trigger mechanism.
- Receiving messages if the channel is open.

X is to stop a local queue and make it unavailable.

R is to refresh the run-time information for this queue from the configuration file, which was updated either by checkpoint requests or MQMT queue configuration. The configuration file (MQFCNFG) contains definitions of the queue manager, channels and queues.

Note: WebSphere MQ configuration is dynamic and should not need to be refreshed during normal operation. If changes have been made to a queue and these changes do not appear to have been recognized by the queue manager, use the (R)efresh option to manually refresh the queue's definition.

Mode The queue process to be operated on, as indicated on screen.

INBOUND Status
Reflects the status of the specified queue. This is normally ACTIVE or IDLE unless the queue inbound has been stopped. If the queue is stopped, DISABLED is also displayed.

OUTBOUND Status
Reflects the status of the specified queue. This is normally ACTIVE or IDLE unless the queue outbound has been stopped. If the queue is stopped, DISABLED is also displayed.

Queuing System Request

Function
The function to be performed, as follows:

S is to start the system queue manager without affecting system resources.

X is to stop the system queue manager without affecting system resources.

Note: WebSphere MQ configuration is dynamic and should not need to be refreshed during normal operation. If changes have been made to a queue and these changes do not appear to have been recognized by the queue manager, use the (R)efresh option to manually refresh the queue's definition.

M toggles the monitor flag. This flag is used to log application requests and their results to the system monitor queue.

Notes:

1. Only local queues can be stopped or started. In order to stop or start a queue that is not local, the queue definition must be updated in the Put-Enabled or Get-Enabled fields.
2. When a local queue is started, any associated triggers will also be started, if the queue depth reflects that messages are present.

   This does not happen when a “Queuing System Request” function is performed. Additionally, any queues that were stopped before the “Queuing System Request” stop function was performed, remain stopped when the corresponding start function is performed.

   Use the Monitor Queue function to check which local queues are stopped.
3. If the queue definition specifies a trigger and a sender channel, then starting a queue triggers the sender program to activate the channel and transmit messages.

Open / Close channel

Selecting 2 on the operations main menu (see Figure 53 on page 135) allows you to open or close communications on an existing channel.

Note: Opening or closing a channel is not the same as starting or stopping the MCA process. See "Communications processes" on page 164 for further information.

Figure 55 shows the first screen to be displayed:

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:21:12               Open / Close Channel                          CIC1
MQWMSC                                                               A002

                           System Information
           System Status    : SYSTEM IS ACTIVE
           Queue Status     : Queuing System is active.
           Channel System   : Channel System is active.
                        Single Channel Request
           Channel Name     :

           Function         :        O=Open , C=Close

           Status           :



                            Channel System Request
           Function         :        O=Open , C=Close


Enter required information.
Enter=Refresh  PF2=Return  PF3=Exit  PF6=Update
```

Figure 55. Open / Close Channel

On this screen, the fields are:

System Information

System Status
Reflects the status of the system. This is normally ACTIVE, unless the system has not been initialized, or has been shut down. When this occurs, the field reads SYSTEM IS SHUTDOWN.

Queue Status
Reflects the status of the queuing system. This is normally ACTIVE, unless the system has not yet been initialized or all the queues have been stopped. When this occurs, the field reads QUEUING SYSTEM IS STOPPED.

Channel System
Reflects the status of the channels. This is normally ACTIVE, unless the system has not yet been initialized or all the channels have been closed. When this occurs, the field reads CHANNEL SYSTEM IS CLOSED.

Single Channel Request

Channel Name
The name of a specific channel to start or stop

Function
The function to be performed:

- O is to open a closed channel.
- C is to close an open channel.

Status Reflects the status of the specified channel. The status can be one of the following:

INACTIVE
Channel is enabled and ready for work.

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
Channel is stopping or a close request has been received.

STOPPED
The channel is disabled, and inactive. A channel in this state can be restarted only by issuing the START CHANNEL command, or using MQMT option 2.2.

Channel System Request

Function
This either opens or closes the channel system.

Note: Opening a channel does not cause a trigger to activate. However, starting the channel's transmission queue does activate a trigger; see Note 3 on page 137.

Reset message sequence number

Selecting 3 on the operations main menu (see Figure 53 on page 135) allows you to reset the message sequence numbers on an existing channel by displaying the screen shown in Figure 56 on page 140:

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:26:47               Reset Channel Message Sequence                CIC1
MQWMMSN                                                              A002

                          System Information
            System Status    : SYSTEM IS ACTIVE
            Queue Status     : QUEUING SYSTEM IS ACTIVE
            Channel Status   : CHANNEL SYSTEM IS ACTIVE
                          Reset Channel Info
            Channel Name     : MQTS.TO.MQ23


            Status           : INACTIVE

            Current Next-MSN : 000000001
            New     Next-MSN :




Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit  PF6=Update
```

Figure 56. Reset channel message sequence

On this screen, the fields are:

System Information

System Status
Reflects the status of the system. This is normally ACTIVE, unless the system has not been initialized, or has been shut down. When this occurs, the field reads SYSTEM IS SHUTDOWN.

Queue Status
Reflects the status of the queuing system. This is normally ACTIVE, unless the system has not yet been initialized or all the queues have been stopped. When this occurs, the field reads QUEUING SYSTEM IS STOPPED.

Channel Status
Reflects the status of the channels. This is normally ACTIVE, unless the system has not yet been initialized or all the channels have been closed. When this occurs, the field reads CHANNEL SYSTEM IS CLOSED.

Reset Channel Info

Channel Name
The name of a specific channel.

Status Reflects the status of the specified channel. The status can be one of the following:

INACTIVE
Channel is enabled and ready for work.

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
Channel is stopping or a close request has been received.

STOPPED
The channel is disabled, and inactive. A channel in this state can be restarted only by issuing the START CHANNEL command, or using MQMT option 2.2.

Current Next-MSN
Displays the message sequence number to be used next.

New Next-MSN
Operator-entered value for message sequence number to be used next.

Note: In order for a channel message sequence number to be reset, the channel must be stopped.

Initialization of system

Selecting 4 on the operations menu allows you to initialize the queuing system by displaying the screen shown in Figure 57.

```
12/27/2008          IBM WebSphere MQ for z/VSE Version 3.0.0         TSMQ300
13:29:47          Initialization / Shutdown of System                CIC1
MQWMSI                                                               A002

                        System Information
           System Status    : SYSTEM IS ACTIVE
           Queue Status     : QUEUING SYSTEM IS ACTIVE
           Channel Status   : CHANNEL SYSTEM IS ACTIVE


           Function         :          I=Initialize, X=Shutdown


           Returned Results :


           System initialized on   12/23/2008 at 10:49:40



Please enter one of the options listed.

Enter=Refresh  PF2=Return  PF3=Exit  PF6=Update
```

Figure 57. Initialization of system

On this screen, the fields are:

System Information

System Status
Reflects the status of the system. This is normally ACTIVE, unless the system has not been initialized, or has been shut down. When this occurs, the field reads SYSTEM IS SHUTDOWN.

Queue Status
Reflects the status of the queuing system. This is normally ACTIVE, unless the system has not yet been initialized or all the queues have been stopped. When this occurs, the field reads QUEUING SYSTEM IS STOPPED.

Channel Status
Reflects the status of the channels. This is normally ACTIVE, unless the system has not yet been initialized or all the channels have been closed. When this occurs, the field reads CHANNEL SYSTEM IS CLOSED.

Function
The function to be performed:

- I is to initialize the system. If the system is initialized, the queue manager is started and all channels and queues opened. Any trigger associated with queues just initialized is also activated if the queue depth is nonzero.
- X is to shut down the system. If the system is shut down, the queue manager is stopped and all the channels closed.

Returned Results
Pressing PF6 with an Initialize function (I) on this screen causes the static system configuration files to be loaded into the CICS for z/VSE dynamic storage. Any error messages or progress messages are displayed in this field.

Note: All outstanding queue maintenance requests must have completed before you perform an initialize or shutdown operation.

Queue maintenance

Selecting 5 on the operations menu allows you either to reset deleted records or physically delete records, by displaying the screen shown in Figure 58 on page 143.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:31:20            Maintain Queue Message Records                   CIC1
MQWMDEL                                                              A002
                            System Information
          System Status    : System is active.
          Queue  Status    : Queuing system is active.
          Channel System   : Channel system is active.
                         Queue Information
          Queue Name       :
          Function         :   A=Delete all
                               D=Delete to date/time exclusive
                               R=Reset from date/time inclusive
          Date (yyyymmdd)  :
          Time (hhmmss)    :

                         Results of Request
          Number Processed :
          Number of Bypass :
          New Last Read QSN:
          Process Time     :

Please enter a Queue name.
Enter=Refresh  PF2=Return  PF3=Exit  PF6=Update
                                          PF12=Retry
```

Figure 58. Maintain Queue Message Records

On this screen, the fields are:

System Information

System Status
Reflects the status of the system. This is normally ACTIVE, unless the system has not been initialized, or has been shut down. When this occurs, the field reads SYSTEM IS SHUTDOWN.

Queue Status
Reflects the status of the queuing system. This is normally ACTIVE, unless the system has not yet been initialized or all the queues have been stopped. When this occurs, the field reads QUEUING SYSTEM IS STOPPED.

Channel System
Reflects the status of the channels. This is normally ACTIVE, unless the system has not yet been initialized or all the channels have been closed. When this occurs, the field reads CHANNEL SYSTEM IS CLOSED.

Queue Information

Queue Name
The name of the local queue on which the function is to be performed.

Function
The function to be performed:

- D is to delete messages that have been logically deleted up to a specified "written" exclusive date and time. For example, given the date and time of 980227230000, specifying "D" deletes all records with a written time prior to 11:00:00 p.m.

  Note: Specifying D does not actually reclaim VSAM space, because record keys are always created in ascending sequence. You are strongly

recommended to read "VSAM file maintenance" on page 183 for information regarding the Delete All function in relation to VSAM files.

- A is to delete all records (logically deleted, or written) and reclaim VSAM space.
- R is to reset all logically deleted records to written status from a specified "deleted" inclusive date and time. For example, given the date and time of 980227230000, specifying "R" resets all delivered messages with delivery time after 10:59:59 p.m.

Date The last date up to which the selected function is to be performed, if applicable.

Time The last time up to which the selected function is to be performed, if applicable.

Queue Information

Number Processed
Number of messages processed

Number of Bypass
Number of messages bypassed

New Last Read QSN
Last read queue sequence number

Process Time
Time to process the request

You use the PF6 (Update) function key to activate the requested function. The function itself is performed by another task, which signals the screen when it is complete. To display the signal, press the Enter key.

The PF12 (Retry) function key is used only if the delete task has ended before finishing the current request, and acts as a new PF6 request.

Note:

1. A Delete or Reset Messages by Date and Time performs the requested function up to the selected date and time, but does not include records with that date and time.
2. If the queue is examined with the browse function, the put time of the last message to be reset should be the value for date and time.
3. The Delete All function purges all records, which include both logically deleted and nondeleted messages.

Once a queue maintenance task is in progress, the task flags the Queue Information entry and logically prevents any other task from accessing this queue. Any attempt to open this queue is rejected with the following message:

```
Queue has xxxx tasks attached. These must be purged.
```

The only action available at this point is to wait and try again later.

Monitor functions

Selecting option 3 (Monitoring) from the master terminal main menu (see Figure 24 on page 80) displays the screen shown in Figure 59 on page 145.

```
12/27/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
13:34:06             *** Monitor Main Menu ***                      CIC1
MQWMMON                                                             A002
                           SYSTEM IS ACTIVE


                             1. Monitor Queue

                             2. Monitor Channel


                        Option:









Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Select  PF2=Return  PF3=Quit
```

Figure 59. Monitor Main Menu

Monitor queues

Selecting 1 on the monitor main menu allows you to monitor the current status of all existing local queues, using the screen shown in Figure 60.

```
12/27/2008           IBM WebSphere MQ for z/VSE Version 3.0.0        TSMQ300
13:35:57                   Monitor Queues                            CIC1
MQWMMOQ                                                              A002
                           QUEUING SYSTEM IS ACTIVE
S QUEUE                          FILE     T INBOUND  OUTBOUND       LR   QDepth
  ANYQ                           MQFI001  N STOPPED  STOPPED         6       10
  SYSTEM.ADMIN.COMMAND.QUEUE     MQFI003  N IDLE     ACTIVE          5        0
  SYSTEM.DEAD.LETTER.QUEUE       MQFERR   N IDLE     IDLE            0        0
  SYSTEM.LOG                     MQFLOG   N IDLE     IDLE            0      134
  SYSTEM.MONITOR                 MQFMON   N IDLE     IDLE            0        0
  VSE1.AIX1.XQ1                  MQFO002  N IDLE     IDLE            4       20
  VSE1.NT1.XQ1                   MQFO002  N IDLE     IDLE            0        2
  WWW.REQ.TYPE7                  MQFI003  Y IDLE     IDLE            3        0









Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit  PF7=Back  PF8=Forward
                             PF9=All   Select or PF10=Detail
```

Figure 60. Monitor queues

This screen displays the current status of all local queues. The displayed fields are:

Queue
Name of the queue.

File CICS FCT DDNAME of a local queue definition.

T Queue type.

N Normal local queue.

Y Transmission local queue.

When PF9 (All) option is selected, additional type values may be displayed. They are:

M Model queue.

Q Manager alias.

A Queue alias.

X Remote queue definition.

Inbound
Status of the inbound process.

ACTIVE
One or more users have the queue open for MQPUT operations.

IDLE No user has the queue open for MQPUT operations.

STOPPED
Queue has been stopped.

MAX At maximum QDepth.

FULL No space.

RECOVERY
For dual queuing.

Outbound
Status of the outbound process.

ACTIVE
One or more users have the queue open for MQGET operations.

IDLE No user has the queue open for MQGET operations.

STOPPED
Queue has been stopped.

RECOVERY
For dual queuing.

LR Last Read: relative record number of the last record on the queue that has been read and processed.

Note: WebSphere MQ messages are logically, rather than physically, deleted from the queue file. **LR** tells you which physical record is prior to the first active record.

QDepth
Estimated queue depth. The approximate number of messages currently on the queue, remaining to be processed.

The QDepth shows the current number of unread messages on a queue, including uncommitted messages being put to the queue. It does not show uncommitted messages currently being retrieved from the queue.

This situation exists because WebSphere MQ for z/VSE updates its internal counters when it detects that an application has syncpointed work. This is only possible on a subsequent MQI call following a SYNCPOINT call. Consequently, there is a brief window when work is actually committed but WebSphere MQ internal counters are not yet updated. This also means that the accuracy of the QDepth is dependant on application design. For more information, see “Syncpoint considerations” on page 943.

Remember also that a QDepth of 0 does not necessarily mean the VSAM file that hosts the queue is empty. Messages that have been retrieved from

the queue using MQGET are actually retained in the queue with a logically deleted status. Such processed messages can only be removed with one of the various reorganization processes.

Note:

1. Pressing the PF9 (All) function key displays an entire list of all queues (local, model, remote, and alias) together with their associated reference.

   If you press this key again, the display lists local queues only.
2. Pressing the PF10 (Detail) function key displays detailed information for this local queue entry.

Monitor queues - detail

An individual queue can be monitored in more detail by placing the cursor anywhere on the line displaying the queue name and pressing PF10, or by placing any character (other than '/') in the S(election) field at the beginning of the line and pressing the Enter key. Either approach activates the Monitor Queues Detail screen as shown in Figure 61.

```
12/27/2008           IBM WebSphere MQ for z/VSE Version 3.0.0        TSMQ300
13:41:05                   Monitor Queues                            CIC1
MQWMMOQ                                                              A002
                           QUEUING SYSTEM IS ACTIVE
                           DETAIL QUEUE INFORMATION
VSE1.AIX1.XQ1
 INBOUND:   STATUS I  ENABLED Y   OPEN Q         0

 OUTBOUND:  STATUS I  ENABLED Y   OPEN Q         0


 BOTH:      FIQ         5    LIQ        24   GETS         0    QDEPTH       20
 TRIGGER:   MAX         1   USED         0   TRAN       PROG. MQPSEND
            CID VSE1.TO.AIX1









Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit  PF10=List
```

Figure 61. Monitor Queues - detail

Monitor channel

Selecting 2 on the monitor main menu (shown in Figure 59 on page 145) allows you to monitor the current status of existing local communications channels, using the screen shown in Figure 62 on page 148.

```
01/13/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
13:42:57                  Monitor Channels                           CIC1
MQWMMOC                                                              A000
                          CHANNEL SYSTEM IS ACTIVE
S CHANNEL              TYPE       MSN       QSN      STATUS
  TEST.MQPUTIL         CLNT         0         0      STOPPED
  VSE7.TO.VSE8         RECV        43        48      INACTIVE
  VSE8.TO.VSE7         SEND       139       145      INACTIVE
  VSE8.TO.WIN1         SEND        12        15      INACTIVE
  WIN1.CLI.VSE8        CLNT         0         0      INACTIVE
  WIN1.TO.VSE7         RECV         0         0      INACTIVE
  WIN1.TO.VSE8         RECV         9       467      INACTIVE









Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit
PF7=Scroll Back    PF8=Scroll Forward   Select or PF10=Detail
```

Figure 62. Monitor channel definitions

This screen displays the current status of local channels.

Channel
Name of the channel.

Type Sender (SEND), Server (SERV), Receiver (RECV), Requester (RQST), or Sevrconn (CLNT)

MSN Last channel message sequence number received or sent.

QSN Queue message sequence number, of the queue name displayed in the **Queue** field.

STATUS
Reflects the status of the specified channel. The status can be one of the following:

INACTIVE
Channel is enabled and ready for work.

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
Channel is stopping or a close request has been received.

STOPPED
The channel is disabled, and inactive. A channel in this state can be restarted only by issuing the START CHANNEL command, or using MQMT option 2.2.

Client (or SVRCONN) channels are a special case. A client channel can have one of the following states:

INACTIVE
Channel is enabled and ready for work.

RUNNING
The channel is currently in use by one or more clients.

STOPPED
The channel is not in use, and is disabled.

Monitor channel - detail

Pressing PF10 (Detail) displays detailed information for a specific channel shown in Figure 63.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:48:59                  Monitor Channels                           CIC1
MQWMMOC                                                              A002
                          CHANNEL SYSTEM IS ACTIVE
                           DETAIL CHANNEL INFORMATION
MQTS.TO.MQ23
           COMMIT          MSN       QSN DATE/TIME
 Sender      BEFORE          0         0 20021126182613

             AFTER           2         3 20021126182613
                     MQTS.MQ23.XQ1









Information displayed.
Enter=Refresh  PF2=Return  PF3=Exit
                                                    PF10=List
```

Figure 63. Monitor channel definitions - detail

This screen displays the channel name, the channel type, and the name of the queue it accesses. The MSN, QSN and time stamp of the last commitment for the BEFORE and AFTER COMMIT fields are also shown.

Message monitoring

For details of message monitoring refer to the WebSphere MQ Monitoring WebSphere MQ manual.

Use is made of the WebSphere MQ display route application, **dspmqrte**, that can be used to configure, generate and put a trace-route message into a queue manager network. Dspmqrte cannot be issued on queue managers before WebSphere MQ

Version 6.0 or on WebSphere MQ for z/VSE. If the first queue manager that the trace route record is put is WebSphere MQ for z/VSE then the the client option **-c** of **dspmqrte** has to be used.

Note:

1. If a trace route message is received on MQSeries for VSE or WebSphere MQ for z/VSE 3.0.0 without the PTF for PM29937, then the put of the message to the queue is rejected because the MQRO_REPORT report option is not supported.
2. The WebSphere MQ for z/VSE sender channel used to send the activity messages to the required queue manager should be defined with a "Get retry delay (secs)" of 10.

Example:

Say you have a network with these 3 queue managers:
- QM1 (WebSphere MQ for z/VSE)
- QM2 (WebSphere MQ for Windows)
- QM3 (WebSphere MQ for z/OS)

To put a trace-route message to QM1, you must perform these steps:

1. Start a command prompt window in the PC running WebSphere MQ for Windows.
2. Set the MQSERVER environment variable with a Server-connection channel defined on the WebSphere MQ for z/VSE server, the constant "TCP", the IPV4 address of the z/VSE system, and the server's listener port number:

   ```
   SET MQSERVER=SYSTEM.AUTO.SVRCONN/TCP/1.2.3.4(1414)
   ```
3. Set the MQCCSID environment variable:

   ```
   SET MQCCSID=819
   ```
4. Issue the dspmqrte command. For example:

   ```
   dspmqrte -c -m QM1 -q Q.ON.QM3.VIA.QM2  -xs 10 -rq Q.ON.QM2
   -rqm QM2 -d yes -v outline -w 10
   ```

   In the example shown, these options are specified:

   -c Request use of client connection to the channel specified in MQSERVER.

   -m Specifies the queue manager to which the client connects.

   -q Specifies the queue where the trace-route message is to be put. This is a remote queue, so the message is forwarded to another queue manager.

   -xs Trace route message expiry. Set to 10 seconds to give time for message to proceed through the network.

   -rq Reply queue where the activity messages are to be put.

   -rqm Queue manager where reply queue resides.

   -d Trace message delivery option.

   -v Display output option. `outline` displays the application name, type of each operation, and operation-specific parameters.

   -w Wait 10 seconds for activity messages to arrive on reply queue.

Controlling queue managers for activity recording

To control whether queue managers are enabled or disabled for activity recording, use the queue manager attribute, ACTIVREC. You can change this by using:

- The MQSC command, ALTER QMGR, specifying the parameter ACTIVREC to change the value of the queue manager attribute.
- The PCF Change Queue Manager (MQCMD_CHANGE_Q_MGR) command with the parameter identifier, MQIA_ACTIVITY_RECORDING.

- The administrator panel Global System Definition Log and Trace Settings. See Figure 64.

Controlling 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. You can change this by using:

- The MQSC command, ALTER QMGR, specifying the parameter ROUTEREC to change the value of the queue manager attribute.
- The PCF Change Queue Manager (MQCMD_CHANGE_Q_MGR) command with the parameter identifier, MQIA_TRACE_ROUTE_RECORDING.
- The administrator panel Global System Definition Log and Trace Settings. See Figure 64.

```
2011/01/17        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
05:11:48                  Global System Definition                  CIC1
MQWMSYS                    Log and Trace Settings                   A004

    Log Settings             Q C       Accounting monitoring
      Informational . . . : Y N          MQI accounting . . . . . : N
      Warning . . . . . . : Y N          Queue accounting . . . . : N
      Error . . . . . . . : Y N          Acc. Conn override . . . : N
      Critical  . . . . . : Y N          Accounting interval  . . : 001800
      Communication . . . : Y N        Statistics monitoring
      Reorganization  . . : Y N          MQI statistics . . . . . : N
      System  . . . . . . : Y N          Queue statistics . . . . : N
                                         Channel statistics . . . : N
    Trace Settings                       Statistics interval  . . : 001800
      MQI calls . . . . . : Y          Online monitoring
      Communication . . . : Y            Queue monitoring . . . . : N
      Reorganization  . . : Y            Channel monitoring . . . : N
      Data conversion . . : Y          Recording
      System  . . . . . . : Y            Activity . . . . . . . . : M
                                         Trace Route  . . . . . . : M

Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 64. Altering recording activity and trace route option

Valid values for recording settings are:

Activity

M Activity reports are generated and sent to the reply queue specified by the originator in the message causing the report. This is the queue manager's initial default value.

Q Activity reports are generated and sent to the local SYSTEM.ADMIN.ACTIVITY.QUEUE.

D Activity reports are not generated.

Trace Route

M Trace-route information is recorded and sent to the destination specified by the originator of the message causing the trace route record. This is the queue manager's initial default value.

Q Trace-route information is recorded and sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

D Trace-route information is not recorded.

Browse function

Selecting option 4 (Browse Queue Records) from the master terminal main menu (see Figure 24 on page 80) displays the screen shown in Figure 65.

```
12/27/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:05:20                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A001

    Object Name: ANYQ
    QSN Number : 00000001       LR-        6, LW-       16, DD-MQFI001
                      Queue Data Record
Record Status : Deleted         PUT date/time  : 20061013093221
Message Size  : 00000200        GET date/time  : 20061013094517
Offset ....+....!....+....!....+....!....+....!....+....!....+....!....+....!
 00000 THIS IS A MESSAGE TEXT
 00070
 00140









Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Hex       PF10=Hdr   PF11=MD
```

Figure 65. Browse Queue Records

This screen shows the content of the message for the specified queue sequence number (QSN) of the chosen object name (queue name). Record status is shown as Written or Deleted along with the associated time stamps.

To browse the queue records, enter the local object name and the QSN of the message of interest. The message on the queue appears in the blank area of the screen, and the message can be manipulated using the function (PF) keys. If the word "ASCII" appears to the right above the scale line, this means the message was ASCII but has been converted to EBCDIC for display purposes. The browse function can only do this if the message's descriptor format is string (MQSTR).

Pressing the PF9 (Hex/Char) key displays the message in hexadecimal or EBCDIC text code.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:58:52                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: Q6.REORG
    QSN Number : 00000001       LR-        0, LW-        4, DD-MQFI003
                   Queue Data Record
Record Status : Written.        PUT date/time  : 20061208124213
Message Size  : 00000200        GET date/time  :                      ASCII
Offset ....+....|....+....|....+....|....+....|....+....|....+....|....+....|
 00000 TEST MSG WITH MQMD VERSION 1 (with ASCII codepage)
       5455245425454245442545544423227676245444266667666222222222222222222222
       45340D37079480D1D4065239FE0108794801339903F450175900000000000000000000
 00070
       2222222222222222222222222222222222222222222222222222222222222222222222
       0000000000000000000000000000000000000000000000000000000000000000000000
 00140                                                              <END>
       2222222222222222222222222222222222222222222222222222234443
       00000000000000000000000000000000000000000000000000000C5E4E

Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Char      PF10=Hdr   PF11=MD
```

Figure 66. Browse Queue Records - Hex display

The PF10 (Txt/Head) key presents detailed WebSphere MQ information for the selected record, and includes channel information if the queue is a transmission queue.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
11:04:50                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: Q6.REORG
    QSN Number : 00000001       LR-        0, LW-        4, DD-MQFI003
                   Internal Header Information
   Original Q - MY.ALIAS.Q6.REORG
   Resolved Q - Q6.REORG
   Resolved QM- TS212.QM.PTHVSEA
   Reply Q    -

   CICS task  -    1256

   Receiver   - PC.TO.VSEA.212
          MSN -         1       Date/Time - 2006/12/08 12:42:13




Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                       PF9=Hex       PF10=Queue PF11=MD
```

Figure 67. Browse Queue Records - Header display

Note: The "Receiver" and "MSN" details are only displayed if the message was written to the queue by the MQ Receiver MCA process. The CICS task is the task that last updated the VSAM record. Consequently, if the message status is

"Written" then this will be the task that put the message and if the status is "deleted", this will be the task that got the message.

PF11 provides information about the message's descriptor (MQMD) data structure.

```
2011/01/25        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
05:50:18                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A002

    Object Name: ANYQ
    QSN Number : 00000001       LR-        0, LW-        1, DD-MQFI001
                      Message Descriptor
 StructID: "MD  "       Version : "00000001"   Report  :X"00000000"
 MsgType : "00000008"   Expiry  : "-0000001"   Feedback: "00000000"
 Encoding: "00000785"   CCSetId : "00001047"   Format  : "MQSTR   "
 Priority: "00000005"   Persist.: "00000001"   PutApplT: "00000010"
 MsgId   :X"C3E2D840D7E3C8E5E2C5C24040404040C73AEDE239D59440"
 Correlid:X"0000000000000000000000000000000000000000000000000000"
 PUTAppl : "TSMQ300 TST4                "
 PUTTime : "21500890"   PUTDate : "20110124"   USERid  : "SIL4        "
 Reply Q : "                                                 "
 Reply QM: "                                                 "



Information displayed.
          5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                       PF9=Hex       PF10=Hdr   PF11=Queue
```

Figure 68. Browse Queue Records - MQMD

Note: The GroupId, MsgSeqNo, Offset and MSGFlags are only displayed if the message's descriptor is a version 2 data structure (MQMD2).

If you browse the System Log file, the PF12 (Help) key appears and can be used to display user action and system action for this message, provided that the system is active.

Note:

1. If the file you are browsing is in the process of being updated by any other WebSphere MQ tasks, this function waits until the completion of those tasks. There can be a delay in the response of the browse function.
2. The Browse utility also has the ability to browse from a queue when the queue manager is not active.
3. Messages can span multiple VSAM records if the message length is greater than the maximum VSAM record length of the file that hosts the queue. The Browse Queue Records function only displays the first VSAM record of the message. In such a case, the browse function displays a warning as follows:

   nnn `bytes of first VSAM record of message displayed`

   where *nnn* is the number of bytes of the message in the first VSAM record.

Administration using the WebSphere MQ Explorer

This information applies to WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform).

WebSphere MQ for Windows and WebSphere MQ for Linux (x86 platform) provide an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using PCF or MQSC commands. You should refer to WebSphere MQ Explorer information on what you can do with the Explorer interface.

The WebSphere MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux (x86 platform), by pointing the WebSphere MQ Explorer at the queue managers and clusters you are interested in.

It allows you to perform tasks, typically associated with setting up and fine tuning the working environment for WebSphere MQ, either locally or remotely within a Windows or Linux (x86 platform) system domain. It monitors the operation of WebSphere MQ servers and provides extensive error detection and recovery functions.

This chapter describes:

What you can do with 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 up 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).
- 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.

Note that not all of these operations are available when using Explorer with a z/VSE queue manager. Operations are enabled where the underlying feature is available with the WebSphere MQ for z/VSE product.

The WebSphere MQ Explorer presents information in a style consistent with that of the WebSphere MQ Eclipse platform.

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

Content View
A Content View is a panel that can display:
- 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

From a Windows or Linux (x86 platform) system, the WebSphere MQ Explorer can connect to all supported queue managers, with these exceptions:

- WebSphere MQ for z/OS queue managers prior to Version 6.0.
- Currently supported MQSeries V2 queue managers except MQSeries for VSE V2.1.2.

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.

Deciding whether to use the WebSphere MQ Explorer

When deciding whether to use the WebSphere MQ Explorer at your installation, bear these points in mind:

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 quotes, or WebSphere MQ will 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. Because 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 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

Before you can use the WebSphere MQ Explorer, you must have these items installed on your computer:

- The WebSphere MQ Eclipse platform (installed as part of WebSphere MQ for Windows or WebSphere MQ for Linux (x86 platform))
- Current service fixes. For WebSphere MQ Explorer to function with a z/VSE queue manager, you must have specific service applied. See “Features” on page 15.

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

Required definitions for administration

Ensure that you have satisfied the following requirements before trying to use the WebSphere MQ Explorer. 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 can be the WebSphere MQ listener or, on UNIX 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 this 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.
4. The system queue, SYSTEM.MQEXPLORER.REPLY.MODEL, must exist.

Cluster membership

If a queue manager is a member 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 this 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.

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, however 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:

- CONNECT authority on the target queue manager object.
- INQUIRE authority on the target queue manager object.
- OUTPUT authority on the queue, SYSTEM.ADMIN.COMMAND.QUEUE.
- DISPLAY and INPUT authority on the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- Authority to perform the action selected.

If a user attempts to perform an operation that they are not authorized to perform, then 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, then authorization failures are generated. If authority events are being recorded, then it is recommended that the user restrict the range of objects that are displayed to those that they have DISPLAY authority to.

Security for connecting to remote queue managers: The 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 specify a nonblank value for the MCAUSER attribute of the channel, or use a security exit, 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.

The default value of the MCAUSER attribute is the local userId. If you specify a nonblank 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.

WebSphere MQ for z/VSE does not support the MCAUSER attribute. Consequently, the Explorer connection runs with the authority of the WMQ z/VSE startup user. For this reason it is advised that client connections are secured with a security exit and possibly an SSL-enabled channel.

Using a security exit: A more flexible approach is to install a security exit on the server-connection channel, typically named SYSTEM.ADMIN.SVRCONN on each queue manager that is to be administered remotely. For information on the supplied security exit, including detailed instructions on setting up and using it, see the WebSphere MQ for Windows Quick Beginnings.

Using SSL security: The 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 the WebSphere MQ Clients book.

Connecting via another queue manager: The WebSphere MQ Explorer allows users to connect to a queue manager via 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 ObjectQMgrName parameter in the object descriptor (MQOD) as the name of the target queue manager. For more information on queue name resolution, see the WebSphere MQ Application Programming Guide.
- 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.

On WebSphere MQ for z/VSE, the authority of the startup user is used to put and get Explorer PCF messages, not the userid specified in the MQMD.

Data conversion

The WebSphere MQ Explorer works by default 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 via 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 the WebSphere MQ Application Programming Reference manual.

In the Global System Definition Communications Settings panel, to ensure that MQ Explorer can connect to WebSphere MQ for z/VSE, you must specify:

```
Cmd Server convert . : Y
```

Using the WebSphere MQ Explorer

This section explains how to use the WebSphere MQ Explorer to:

- Show or hide queue managers.
- Use the WebSphere MQ Taskbar application (Windows only).
- Use the WebSphere MQ alert monitor application (Windows only).

Showing and hiding queue managers and clusters

The WebSphere MQ Explorer can display more than one queue manager at a time. The Show/Hide Queue Manager panel (selectable from the context menu for the Queue Managers tree node) allows you to choose whether you display information on 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. Fill in 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, and allows you 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.

Using the WebSphere MQ Taskbar application (Windows only)

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 these meanings:

Green Healthy; 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

When you click on the icon with your right mouse button, a context menu is displayed. From this menu, select the WebSphere MQ Explorer option to bring up the WebSphere MQ Explorer.

Using 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.

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.

Administration via a web browser

Although WebSphere MQ for VSE and its administration programs run in a CICS environment, administration of WebSphere MQ objects can be performed via a web browser.

Administering WebSphere MQ in this way takes advantage of the CICS Web Support (CWS) feature of CICS Transaction Server for z/VSE 1.1.1.

This section provides an overview of the CWS feature, describes the WebSphere MQ modules that facilitate administration from a web browser, and explains how to use CWS with WebSphere MQ.

CICS Web Support

CWS is a 2-tier connector solution based on HTTP/HTML. It facilitates connectivity between a web browser and z/VSE CICS TS applications. For WebSphere MQ for z/VSE, this means connectivity between a web browser and the WebSphere MQ administration programs.

The WebSphere MQ administration programs are 3270-based CICS applications. CWS provides a 3270 bridge solution which allows access to 3270-based CICS applications without 3270 terminals. Administering WebSphere MQ from a web browser exploits the 3270 bridge feature of CWS.

The operating environment and minimum prerequisites for CWS are:

- z/VSE 3.1 or later.
- CICS Transaction Server 1.1.1 or later.
- TCP/IP for z/VSE 1.5F (or equivalent) or later.
- Language environment for z/VSE 1.4.4 or later.

In addition, to administer WebSphere MQ from a web browser, CWS must be correctly implemented, and the CICS/TS region hosting WebSphere MQ must be properly configured for CWS use. To this end, you should refer to the following manuals:

- CICS Transaction Server for z/VSE Enhancement Guide (SC34-5763).
- CICS Transaction Server for z/VSE Internet Guide (SC34-5765).
- CICS Transaction Server for z/VSE CICS Web Support (SG24-5997-00).

Note that these manuals pertain to CICS Transaction Server for z/VSE V1.1.1. WebSphere MQ administration via a web browser is not possible with CICS for z/VSE or CICS/TS releases prior to V1.1.1.

CWS WebSphere MQ modules

WebSphere MQ for VSE provides HTML source that corresponds to each of its administration screens. The source is provided in the WebSphere MQ installation library in member MQHTML.Z.

In addition, WebSphere MQ for z/VSE provides a 3270/HTML converter program called MQPCWS.

Each of these modules is described below.

HTML source file

The MQHTML.Z file contains HTML source generated by an assembly of the WebSphere MQ MAP programs using the TYPE=TEMPLATE parameter of the DFHMSD macro. This is the standard method for generating HTML source from CICS Basic Mapping Support (BMS) programs.

The MQHTML.Z file contains a series of LIBRARIAN CATALOG statements followed by HTML source relevant to a particular administration screen. It is intended to be used as input to the Librarian utility program (LIBR) to create individual HTML source members for each WebSphere MQ administration screen.

The following JCL provides an example of how the MQHTML.Z file might be loaded into a sublibrary for CWS use:

```
* $$ JOB MQCWSLD,CLASS=0
* $$ LST CLASS=A,DISP=H
// JOB
// EXEC LIBR
ACC S=library.dfhdoc
* $$ SLI MEM=MQHTML.Z,S=PRD2.WMQZVSE
/*
/&
* $$ EOJ
```

where `library.dfhdoc` should be replaced with a sublibrary name intended for CWS use. If WebSphere MQ is installed in a sublibrary other than PRD2.WMQZVSE, the `* $$ SLI` card should also be changed.

File MQJCWS.Z contains this same JCL, and is available in the WebSphere MQ for z/VSE installation library.

Once the HTML source has been loaded into an appropriate documents sublibrary (that is, a library configured for CWS), the HTML source is ready for use.

CWS converter program

Converter programs are optional programs used to support the operation of CWS. Essentially, a converter is an exit program called before and after the 3270/HTML data flow to and from a CICS TS program.

A converter must provide two functions:

- Decode is used before the CICS TS program is called. It can:
  - Use the data from the Web browser to build the communication area in the format expected by the CICS TS program.

  - Supply the lengths of the input and output data in the CICS TS program communication area.
  - Perform administrative tasks related to the response.
- Encode is used after the CICS TS program has been called. It can:
  - Use the data from the CICS TS program to build the HTTP response and HTTP response headers.
  - Perform administrative tasks related to the response.

On some browsers the web page layout generated from the provided HTML source may require improvement. For this reason, WebSphere MQ for z/VSE provides a sample 3270/HTML converter program (MQPCWS) which is intended to improve the overall appearance of the WebSphere MQ administration screens in a web browser.

The converter program, MQPCWS, is provided as a sample only, and is available in the WebSphere MQ installation sublibrary as both an executable and a COBOL source file.

Using CWS with WebSphere MQ

Once the CWS environment has been implemented and the WebSphere MQ HTML files have been loaded into an appropriate CWS documents sublibrary, WebSphere MQ can be administered from a web browser.

WebSphere MQ does not need to be active to administer WebSphere MQ from a browser since the activation of the system is itself an administration function that can be performed from the administration screens. However, to administer WebSphere MQ from a browser, TCP/IP services and the CICS TS region hosting WebSphere MQ must be active.

WebSphere MQ administration transactions are started from a browser by requesting an appropriate Universal Resource Locator (URL). For example:

```
http://n.n.n.n:pppp/cics/cwba/dfhwbtta/TTTT
```

where `n.n.n.n` is the IP address of z/VSE system running the CICS TS system that hosts WebSphere MQ, pppp is the CICS listener port number for CWS, and TTTT is the WebSphere MQ administration transaction identifier (for example MQMT).

Valid WebSphere MQ transaction identifiers are listed in section “Master Terminal transactions” on page 80.

As already suggested, the appearance of the administration screens may be improved using the supplied CWS converter program, MQPCWS. A converter program can be introduced by changing the URL path. For example:

```
http://n.n.n.n:pppp/mqpcws/cwba/dfhwbtta/TTTT
```

Activating an WebSphere MQ administration transaction this way will ensure that the MQPCWS converter program is called to encode and decode the 3270/HTML data flow.

Communications processes

WebSphere MQ uses the Message Channel Agent (MCA) programs and TCP/IP Listener program for its communications.

The MCA process:

- Runs as a separate CICS task connected to the remote WebSphere MQ using APPC or TCP/IP protocol.
- Starts automatically in response to other system activity, or when a message is placed on a transmission queue.
- Starts the WebSphere MQ server when initial client requests are received.
- Can be stopped from the operations main menu.
- When processing a channel, ensures SSL services are activated if required.

Select 2 from the operations main menu to control the channels. See "Open / Close channel" on page 138 for further information.

The WebSphere MQ listener process:

- Establishes itself as a TCP/IP "listener" process by binding itself to a port number configured in the global system definition.
- Runs as a separate CICS task waiting for remote TCP/IP connection requests.
- Starts the receiver MCA when connection requests are received.
- Ends when WebSphere MQ is shut down.

Message persistence

Message persistence is a field of the MQMD data structure, that accompanies all MQ messages.

Message persistence indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT
MQPER_NOT_PERSISTENT
MQPER_PERSISTENCE_AS_Q_DEF

MQPER_PERSISTENT

Message is persistent. This means that the message survives system failures and restarts of the queue manager. Once the message has been put, and the putter's unit of work committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the getter's unit of work committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism is used to hold the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on temporary dynamic queues.

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT

Message is not persistent. This means that the message does not normally survive system failures or restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during restart of the queue manager.

Non-persistent messages can be placed on temporary dynamic, permanent dynamic and predefined queues.

MQPER_PERSISTENCE_AS_Q_DEF

Message has default persistence.

On z/VSE, default persistence cannot be specified as part of the queue definition. Instead, predefined and permanent dynamic queues have a default persistence of PERSISTENT. Temporary dynamic queues have a default persistence of NOT_PERSISTENT.

When a message is placed on a queue with MQPER_PERSISTENCE_AS_Q_DEF, the relevant persistence is defaulted accordingly for the message.

Message expiry

By default, messages placed on a queue have an unlimited 'lifetime'. This is the amount of time a message will stay on a queue before it is discarded by the queue manager, assuming it is not retrieved by an application.

Optionally, messages can be placed on a queue with a limited 'lifetime'. The amount of time a message will remain on a queue before it is discarded by the queue manager is called the message 'expiry'. A message's expiry is specified in the message descriptor (MQMD) data structure when the message is placed on a queue.

The message expiry is period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

The value is decremented to reflect the time the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It may also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the Expiry field represents the amount of the original expiry time that still remains.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the Expiry field of the message descriptor after a successful MQGET call is either greater than zero, or the special value MQEI_UNLIMITED.

If a message is put on a remote queue, the message may expire (and be discarded) whilst it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the MQRO_EXPIRATION_* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (usually the queue specified by the ReplyToQ field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note:

1. If a message is put with an Expiry time of zero, the MQPUT or MQPUT1 call fails with reason code MQRC_EXPIRY_ERROR; no report message is generated in this case.
2. Since a message whose expiry time has elapsed may not actually be discarded until later, there may be messages on a queue that have passed their expiry time, and which are not therefore eligible for retrieval. These messages nevertheless count towards the number of messages on the queue for all purposes, including depth triggering.
3. An expiration report is generated, if requested, when the message is actually discarded, not when it becomes eligible for discarding.
4. Discarding of an expired message, and the generation of an expiration report if requested, are never part of the application's unit of work, even if the message was scheduled for discarding as a result of an MQGET call operating within a unit of work.
5. If a nearly-expired message is retrieved by an MQGET call within a unit of work, and the unit of work is subsequently backed out, the message may become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with MQGMO_LOCK, the message may become eligible to be discarded before it can be retrieved by an MQGET call with MQGMO_MSG_UNDER_CURSOR; reason code MQRC_NO_MSG_UNDER_CURSOR is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:
   - Copy the remaining expiry time from the request message to the reply message.
   - Set the expiry time in the reply message to an explicit value greater than zero.
   - Set the expiry time in the reply message to MQEI_UNLIMITED.

   The action to take depends on the design of the application suite. However, the default action for putting messages to a dead-letter (undelivered-message) queue should be to preserve the remaining expiry time of the message, and to continue to decrement it.

8. Expiry report messages are always generated with MQEI_UNLIMITED.
9. A message (normally on a transmission queue) which has a Format name of MQFMT_XMIT_Q_HEADER has a second message descriptor within the MQXQH. It therefore has two Expiry fields associated with it. The following additional points should be noted in this case:
   - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two Expiry fields to be the same as that specified by the application.
   - If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be MQFMT_XMIT_Q_HEADER (but the queue manager does not enforce this). In this case the application need not set the values of these two Expiry fields to be the same. (The queue manager does not check that the Expiry field within the MQXQH contains a valid value, or even that the message data is long enough to include it.)
   - When a message with a Format name of MQFMT_XMIT_Q_HEADER is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements both these Expiry fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the Expiry field in the MQXQH.
   - The queue manager uses the Expiry field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
   - If the initial values of the two Expiry fields were different, it is therefore possible for the Expiry time in the separate message descriptor when the message is retrieved to be greater than zero (so the message is not eligible for discarding), while the time according to the Expiry field in the MQXQH has elapsed. In this case the Expiry field in the MQXQH is set to zero.

Viewing error logs

WebSphere MQ error messages, and other system messages, are placed on the following queues:

SYSTEM.LOG
: All WebSphere MQ generated error messages are written to this queue. If SYSTEM.LOG is not defined, or if WebSphere MQ cannot successfully write to it, the error messages are logged to CSMT and may be viewed using standard system utilities. The CSMT redirection parameter is an active toggle, and can be set in the global system definition.

SYSTEM.DEAD.LETTER.QUEUE
: Is the WebSphere MQ dead letter queue. Messages that cannot be queued to their specified destination are queued here.

SYSTEM.MONITOR
: API monitor queue used to log all application requests and their results. This is primarily for problem determination purposes.

Note:

1. The names listed for these queues are the default names, but you can redefine the actual queue names through the global system definition screen.

2. You can view the messages written to these queues using the WebSphere MQ browse queue function (see “Browse function” on page 152).
3. The messages included in the SYSTEM.LOG can be controlled using the 'Log and Trace Settings' screen. Refer to “Queue Manager Log and Trace Settings” on page 88.

Chapter 5. Utilities and interfaces

WebSphere MQ for z/VSE is supplied with various utility functions, which are:

- The WebSphere MQ System Administration Control Interface - see "System Administration Control Interface"
- Batch utilities - see "Batch utilities" on page 173
- The batch interface - see "Using the batch interface" on page 178
- Utilities for reclaiming VSAM file space - see "VSAM file maintenance" on page 183
- The WebSphere MQ-CICS bridge - see "WebSphere MQ-CICS Bridge" on page 186

System Administration Control Interface

The WebSphere MQ System Administration Control Interface allows a limited number of system administration functions to be performed using programs.

The System Administration Control Interface has:

- A transactional interface (MQCL).
- A programmable interface (MQPCMD).

Transactional interface (MQCL)

MQCL is the command-line interface to the MQPCMD program. It allows you selectively to:

- Stop, start, or query Inbound and Outbound queues.
- Open, close, or query a channel.

The command may be entered on cleared CICS terminal or on the VSE console as a command to the CICS partition.

Running the MQCL transaction without parameters generates the following display:

```
MQCL
MQCL is a command line interface which allows queues and
channels to be selectively opened and closed. It has the
flexibility to open and close in inbound, outbound or both
directions. The syntax format is:

MQCL FF NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

where NN...NN = Queue or Channel name
      FF = one of following function codes
CHANNEL - Type       Start  Inact  Stop  Query
          SENDER      SS     IS     PS    QS
          RECEIVER    SR     IR     PR    QR
          SERVER      SV     IV     PV    QV
          REQUESTER   SQ     IQ     PQ    QQ
          SVRCONN     SC     IC     PC    QC

QUEUE   - Direction  Start  Stop   Query
          INBOUND     SI     XI     QI
          OUTBOUND    SO     XO     QO
          BOTH        SB     XB     QB
QUEUE Depth                         QD
MQM001090 Command Line Program invalid syntax.
```

Figure 69. MQCL syntax display

To query a channel, use FF value of QS, QR, Qv, QQ or QC. For example, if you want to query a sender channel called VSE9.IP.SOL7, use the following command:

```
MQCL QS VSE9.IP.SOL7
```

If successful, this command will generate a display similar to the following:

```
MQM001000 CHANNEL STATUS: INACTIVE      (SDR)      VSE9.IP.SOL7
```

When stopping channels using MQCL, you can request that the channel is stopped and placed in INACTIVE state by using function codes from the Inact column. To stop a channel and place it in STOPPED state, use one of the function codes in the Stop column.

A message is sent to the activated terminal for every activation of this transaction. The original command is redisplayed if entered in a CICS session, eg

```
MQCL QB IBM.REPLY.QUEUE
   MQM001000 QUEUE STATUS: IN=IDLE    ,OUT=IDLE    IBM.REPLY.QUEUE
```

This message has a message code of MQM001000 for completed messages, or an error code of MQM001090 for any that did not complete properly. The text that follows explains the exact results.

Programmable interface (MQPCMD)

The programmable interface uses an EXEC CICS LINK PROGRAM (MQPCMD). MQPCMD accepts a fixed format COMMAREA which specifies the type of request and a status response area.

The supplied copybook MQICMD.C describes this area.

```
*-----------------------------------------------------------------*
* - BEGIN -     ***  COPYBOOK: MQICMD    ***           - BEGIN - *
*-----------------------------------------------------------------*
*-----------------------------------------------------------------*
*  COMMAND LINE COPYBOOK                                          *
*-----------------------------------------------------------------*
```

```
01  MQI-COMMAND-LINE.
    02  MQI-CMD-PASSED-AREA.
       05  MQI-CMD-TRANS-ID           PIC  X(4) VALUE 'MQCL'.
       05  FILLER                     PIC  X   VALUE SPACE.
       05  MQI-CMD-FUNCTION           PIC  XX  VALUE SPACE.
           88 MQI-CMD-FUNCTION-OK     VALUE 'CR', 'OR', 'QR',
                                           'SR', 'IR', 'PR',
                                           'CS', 'OS', 'QS',
                                           'SS', 'IS', 'PS',
                                      'QV', 'SV', 'IV', 'PV',
                                      'QQ', 'SQ', 'IQ', 'PQ',
                                           'CC', 'OC', 'QC',
                                           'SC', 'IC', 'PC',
                                           'QB', 'QI', 'QO',
                                           'QD',
                                           'XB', 'XI', 'XO',
                                           'SB', 'SI', 'SO'.
           88 MQI-CMD-FUNC-CHANNEL    VALUE 'CR', 'OR', 'QR',
                                           'SR', 'IR', 'PR',
                                           'CS', 'OS', 'QS',
                                           'SS', 'IS', 'PS',
                                      'QV', 'SV', 'IV', 'PV',
                                      'QQ', 'SQ', 'IQ', 'PQ',
                                           'CC', 'OC', 'QC',
                                           'SC', 'IC', 'PC'.
           88 MQI-CMD-CLOSE-CHANNEL   VALUE 'CR', 'CS', 'CC',
                                           'IR', 'IS', 'IC',
                                           'IV', 'IQ',
                                           'PR', 'PS', 'PC',
                                           'PV', 'PQ'.
           88 MQI-CMD-OPEN-CHANNEL    VALUE 'OR', 'OS', 'OC',
                                           'SR', 'SS', 'SC',
                                           'SV', 'SQ'.
           88 MQI-CMD-QUERY-CHANNEL   VALUE 'QR', 'QS', 'QC',
                                           'QV', 'QQ'.
           88 MQI-CMD-CHANNEL-SEND    VALUE 'CS', 'OS', 'QS',
                                           'SS', 'IS', 'PS'.
           88 MQI-CMD-CHANNEL-RECVR   VALUE 'CR', 'OR', 'QR',
                                           'SR', 'IR', 'PR'.
           88 MQI-CMD-CHANNEL-CLIENT  VALUE 'CC', 'OC', 'QC',
                                           'SC', 'IC', 'PC'.
           88 MQI-CMD-CHANNEL-SVR     VALUE 'SV', 'IV', 'PV',
                                           'QV'.
           88 MQI-CMD-CHANNEL-RQSTR   VALUE 'SQ', 'IQ', 'PQ',
                                           'QQ'.
           88 MQI-CMD-INACTIVE-STOP   VALUE 'IS', 'IR', 'IC',
                                           'IV', 'IQ'.

           88 MQI-CMD-FUNC-QUEUE      VALUE 'XB', 'XI', 'XO',
                                           'SB', 'SI', 'SO',
                                           'QB', 'QI', 'QO',
                                           'QD'.
           88 MQI-CMD-STOP-QUEUE      VALUE 'XB', 'XI', 'XO'.
           88 MQI-CMD-STOP-Q-INBOUND  VALUE 'XI'.
           88 MQI-CMD-STOP-Q-OUTBOUND VALUE 'XO'.
           88 MQI-CMD-STOP-Q-BOTH     VALUE 'XB'.
           88 MQI-CMD-START-QUEUE     VALUE 'SB', 'SI', 'SO'.
           88 MQI-CMD-START-Q-INBOUND VALUE 'SI'.
           88 MQI-CMD-START-Q-OUTBOUND VALUE 'SO'.
           88 MQI-CMD-START-Q-BOTH    VALUE 'SB'.
           88 MQI-CMD-QUERY           VALUE 'QB', 'QI', 'QO',
                                           'QS', 'QR', 'QC',
                                           'QV', 'QQ', 'QD'.
           88 MQI-CMD-QUERY-QUEUE     VALUE 'QB', 'QI', 'QO',
                                           'QD'.
           88 MQI-CMD-QUERY-Q-INBOUND VALUE 'QI'.
```

```
                88 MQI-CMD-QUERY-Q-OUTBOUND VALUE 'QO'.
                88 MQI-CMD-QUERY-Q-BOTH     VALUE 'QB'.
                88 MQI-CMD-QUERY-Q-DEPTH    VALUE 'QD'.

                 88 MQI-CMD-Q-IN             VALUE 'SI', 'XI'.
                 88 MQI-CMD-Q-OUT            VALUE 'SO', 'XO'.
                 88 MQI-CMD-Q-BOTH           VALUE 'SB', 'XB'.

             05  FILLER                   PIC  X   VALUE SPACE.
             05  MQI-CMD-QUEUE-NAME       PIC  X(48) VALUE SPACES.
             05  MQI-CMD-CHANNEL-NAME     REDEFINES
                 MQI-CMD-QUEUE-NAME      PIC  X(48).

      *-------values returned when LINKed
          02  MQI-CMD-RETURNED-AREA.
             05  MQI-CMD-RC               PIC S9(4) COMP VALUE ZERO.
              88 MQI-CMD-RC-OK                          VALUE ZERO.
              88 MQI-CMD-RC-DUPLICATE-FUNC              VALUE +4.
              88 MQI-CMD-RC-NAME-INVALID                VALUE +10.
              88 MQI-CMD-RC-SYS-NOT-ACTIVE              VALUE +80.
              88 MQI-CMD-RC-ERRORS                      VALUE +90.
              88 MQI-CMD-RC-HELP                        VALUE +99.

             05 MQI-CMD-ERROR-LINE.
               10  MQI-CMD-ERROR-PREFIX   PIC  XXX   VALUE 'MQM'.
               10  MQI-CMD-ERROR-CODE     PIC  X(6)  VALUE SPACES.
               10  FILLER                 PIC  X     VALUE SPACE.
               10  MQI-CMD-ERROR-TEXT     PIC  X(40) VALUE SPACES.
               10  FILLER                 PIC  X     VALUE SPACE.
               10  MQI-CMD-ERROR-NAME     PIC  X(48) VALUE SPACES.
             05  MQI-CMD-QUERY-STATES.
               10  MQI-CMD-QUERY-RC-INBOUND    PIC X(8) VALUE SPACES.
                   88 MQI-CMD-QUERY-RC-IN-MAX      VALUE 'MAX     '.
                   88 MQI-CMD-QUERY-RC-IN-FULL     VALUE 'FULL    '.
                   88 MQI-CMD-QUERY-RC-IN-ERRORED  VALUE 'ERRORED '.
                   88 MQI-CMD-QUERY-RC-IN-IDLE     VALUE 'IDLE    '.
                   88 MQI-CMD-QUERY-RC-IN-ACTIVE   VALUE 'ACTIVE  '.
                   88 MQI-CMD-QUERY-RC-IN-INHIBIT  VALUE 'INHIBIT '.
                   88 MQI-CMD-QUERY-RC-IN-RECOVER  VALUE 'RECOVER '.
                   88 MQI-CMD-QUERY-RC-IN-STOPPED  VALUE 'STOPPED '.
               10  MQI-CMD-QUERY-RC-CHANNEL
                   REDEFINES  MQI-CMD-QUERY-RC-INBOUND  PIC X(8).
                   88 MQI-CMD-QUERY-RC-INACTIVE    VALUE 'INACTIVE'.
                   88 MQI-CMD-QUERY-RC-BINDING     VALUE 'BINDING '.
                   88 MQI-CMD-QUERY-RC-STARTING    VALUE 'STARTING'.
                   88 MQI-CMD-QUERY-RC-RUNNING     VALUE 'RUNNING '.
                   88 MQI-CMD-QUERY-RC-STOPPING    VALUE 'STOPPING'.
                   88 MQI-CMD-QUERY-RC-STOPPED     VALUE 'STOPPED '.
                   88 MQI-CMD-QUERY-RC-RETRYING    VALUE 'RETRYING'.
                   88 MQI-CMD-QUERY-RC-UNKNOWN     VALUE 'UNKNOWN '.
               10  FILLER REDEFINES
                   MQI-CMD-QUERY-RC-INBOUND.
                   15 MQI-CMD-QUERY-RC-DEPTH   PIC S9(8) COMP.
                   15 FILLER                   PIC X(4).
               10  MQI-CMD-QUERY-RC-OUTBOUND   PIC X(8) VALUE SPACES.
                   88 MQI-CMD-QUERY-RC-OUT-ERRORED VALUE 'ERRORED '.
                   88 MQI-CMD-QUERY-RC-OUT-IDLE    VALUE 'IDLE    '.
                   88 MQI-CMD-QUERY-RC-OUT-ACTIVE  VALUE 'ACTIVE  '.
                   88 MQI-CMD-QUERY-RC-OUT-INHIBIT VALUE 'INHIBIT '.
                   88 MQI-CMD-QUERY-RC-OUT-RECOVER VALUE 'RECOVER '.
                   88 MQI-CMD-QUERY-RC-OUT-STOPPED VALUE 'STOPPED '.

      *---------------------------------------------------------------*
      * - END -        ***  COPYBOOK: MQICMD    ***           - END - *
      *---------------------------------------------------------------*
```

The following sample program is an example of how to use MQICMD.C in a CICS application program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TESTCMD.
AUTHOR. IBM.
DATE-COMPILED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  WS-QCLOSED  PIC X(20) VALUE 'Queue closed OK.'.
01  WS-QNCLOSED PIC X(20) VALUE 'Queue not closed.'.

    COPY MQICMD.

PROCEDURE DIVISION.

0000-MAIN-LINE.
    MOVE 'ANYQ'               TO MQI-CMD-QUEUE-NAME.
    SET  MQI-CMD-STOP-Q-BOTH   TO TRUE.
    EXEC CICS LINK
                  PROGRAM('MQPCMD')
                  COMMAREA(MQI-COMMAND-LINE)
                  LENGTH(LENGTH OF MQI-COMMAND-LINE)
    END-EXEC.
    EVALUATE TRUE
       WHEN MQI-CMD-RC-OK
          EXEC CICS WRITE OPERATOR
                          TEXT(WS-QCLOSED)
                          TEXTLENGTH(LENGTH OF WS-QCLOSED)
          END-EXEC
       WHEN OTHER
          EXEC CICS WRITE OPERATOR
                          TEXT(WS-QNCLOSED)
                          TEXTLENGTH(LENGTH OF WS-QNCLOSED)
          END-EXEC
    END-EVALUATE.

0000-RETURN.
    EXEC CICS RETURN
    END-EXEC.

    GOBACK.
```

Batch utilities

WebSphere MQ for z/VSE provides several batch utility programs. These are programs that can be run from a batch partition to manipulate or report on WebSphere MQ resources.

Batch utilities include:
- MQPUTIL
- MQPEXCIC

MQPUTIL program

The PRD2.WMQZVSE library contains all the sample code and JCL, including the MQJUTILY.Z example background batch job.

MQJUTILY.Z contains the MQPUTIL program, which performs the following functions:

- Prints the system, queue, and channel definitions from a configuration file.

- Prints the SYSTEM.LOG file in a formatted report.
- Prints the SYSTEM.MONITOR queue in a formatted report.
- Updates all channels with a new starting MSN.
- Updates a configuration file for dual queues. It makes all dual queues into a primary queue.
- Prints new Help Facility error information.
- Prints code pages recognized by WebSphere MQ for z/VSE.
- Updates system configuration constants.

The MQPUTIL program uses the DLBL filename CONFIG for the WebSphere MQ configuration file, and the PRINT LOG commands require the DLBL filename INLOG for the VSAM file containing the system log queue. MQPUTIL uses logical unit SYSIPT to read requests and writes output to SYS005, which should be assigned to a PRINTER device. The MQPUTIL program uses the following general syntax:

Table 8. MQPUTIL program general syntax

Column	Content
1 to 5	Command name
6	Space
7 to 18	Subcommand
19	Space
20...	Arguments

The supplied MQPUTIL program has four commands, which are:
- "PRINT"
- "RESET" on page 175
- "DUALQ" on page 175
- "UPDATE" on page 176

Note: You must run the MQPUTIL program once for each command that you require.

PRINT

PRINT has seven options:

CONFIG
Prints the full WebSphere MQ configuration specified by the CONFIG DLBL.

LOG Prints the system log in a formatted report specified by the INLOG DLBL.

LOG FROMQ system.log
Prints active messages only from the system log in a formatted report. The LOG FROMQ option removes messages from the system log as it generates its report. It does not include processed messages, those already logically deleted, in the report.

Since the system log name is configurable, the name of the system log queue must be provided as an parameter to the LOG FROMQ option.

The LOG FROMQ option uses the WebSphere MQ for z/VSE Batch Interface (see "Using the batch interface" on page 178). This means the z/VSE queue manager, and the batch interface must be active when this option is used. If the Batch Interface uses an identifier other than

MQBISERV, the JCL to run MQPUTIL will also require a // SETPARM MQBISRV card to identify the name of the appropriate batch interface.

In the PRINT LOG reports, each formatted message starts with a line of hyphens (-) for informational or warning messages, pluses (+) for error messages, and asterisks (*) for critical messages. This is to aid readability.

MESSAGES
Prints a Help Facility resolution report using the configuration file specified by the CONFIG DLBL.

MONITOR
Prints all messages in the monitor queue in a formatted report using the INLOG DLBL which contains the fileid of the VSAM file specified for the monitor queue.

MONITOR FROMQ *monitor.queue*
Prints active messages only from the monitor queue in a formatted report. The MONITOR FROMQ option removes messages from the monitor queue as it generates its report. It does not include processed messages, those already logically deleted, in the report. Since the monitor queue name is configurable, the name of the monitor queue must be provided as a parameter to the MONITOR FROMQ option.

The MONITOR FROMQ option uses the WebSphere MQ for z/VSE Batch Interface. This means the z/VSE queue manager, and the batch interface must be active when this option is used. If the Batch Interface uses an identifier other than MQBISERV, the JCL to run MQPUTIL will also require a // SETPARM MQBISRV card to identify the name of the appropriate batch interface.

CCSID
For WebSphere MQ for z/VSE data conversion, the source and target code page numbers are first checked against an internal list of valid code pages. If the code page does not exist in this list, it is then checked against any user code page defined by the administrator Code Page Definitions panels. If the code page has not been defined as a user code page, then the data conversion will be rejected. The PRINT CCSID option will allow you to check if a code page will be accepted for data conversion.

Note: The fact that the code page is valid does not mean that the data conversion will work. WebSphere MQ for z/VSE uses Language Environment/VSE services to do the conversion. Please refer to “Code page conversion” on page 1011 for requirements.

RESET

MSN nnnnnnnn
Resets all channel numbers to nnnnnnnn and checkpoint values to zero.

DUALQ

DUALQ has the following option:

TAKEOVER dual_queue_name
Allows the dual queue specified to become the primary queue, using the following process:

1. The configuration file points to the cluster hosting the dual queue instead of to the cluster hosting the primary queue.
2. All message headers in the dual queue are modified to contain the name of the primary queue instead of the name of the dual queue.

This command may be used when a local queue becomes unavailable, for example, when input or output errors occur, and a dual queue has been defined.

Note: You are recommended to backup the configuration file, using the VSAM REPRO command, before using this command, because the file will be changed. The configuration file can be restored when you have repaired the failure.

Refer to the sample JCL stream in Appendix D, "Sample JCL and programs," on page 977.

UPDATE

The MQPUTIL utility provides a means to perform an upgrade of the queue manager's configuration constants. This upgrade is usually handled by the MQSU transaction during installation or following the application of product maintenance. See "Starting WebSphere MQ" on page 1067 for more information about the MQSU transaction.

In regard to maintenance, whenever an WebSphere MQ for z/VSE panel or message text is changed, an updated SYSIN.Z file is included in the relevant PTF. This requires the batch job MQJSETUP to be run and then the CICS transaction MQSU to be run before restarting WebSphere MQ for z/VSE. For convenience, so that these maintenance steps can be managed exclusively from batch, you can use the UPDATE command.

UPDATE has four options:

UPDATE
Updates the file CONFIG with data read from SYSIPT that follows this command.

UPDATE UPPERCASE
Updates the file CONFIG with data read from SYSIPT that follows this command. The UPPERCASE keyword forces all panel and message text to be folded to uppercase.

UPDATE FROM MQFSSET
Updates the file CONFIG with data read from file MQFSSET that has been loaded with the SYSIN.Z data. Refer to MQJSETUP.Z

UPDATE FROM MQFSSET UPPERCASE
Updates the file CONFIG with data read from file MQFSSET that has been loaded with the SYSIN.Z data. Refer to MQJSETUP.Z. The UPPERCASE keyword forces all panel and message text to be folded to uppercase.

These parameters contained in SYSIN.Z can be changed without having to edit the SYSIN.Z by adding SET statements following the UPDATE command:

DATE The format of the data that appears in the administrator panels and in the SYSTEM.LOG messages.

For example:

```
SET DATE nn
```

where *nn* has one of these values:

01 for MM/DD/YYYY
02 for YYYY/MM/DD
03 for YYYY/DD/MM
04 for YYYY/DDD
05 for DD/MM/YYYY

QM-SUBSYSID
The 4-character subsystem ID that can be used in building the resource name used for WebSphere MQ for z/VSE security.

For example:

```
SET   QM-SUBSYSID   VSE1
```

QM-STATUS-SECURITY
Used to ENABLE WebSphere MQ for z/VSE security.

For example:

```
SET QM-STATUS-SECURITY  ENABLED
```

QM-AUDIT-SECURITY
This is not currently used by WebSphere MQ for z/VSE security.

QM-COMPAT-MODE
Can be set to ENABLED or DISABLED.

For example, if you plan to use POWER® SLI to include SYSIN.Z data to be read from SYSIPT:

```
// DLBL CONFIG,'WMQZVSE.mqfcnfg',,VSAM,CAT=VSESPUC
// LIBDEF PHASE,SEARCH=(PRD2.WMQZVSE,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPUTIL,SIZE=MQPUTIL
UPDATE
* $$ SLI MEM=SYSIN.Z,S=PRD2.WMQZVSE
/*
```

Alternatively, if you plan to use MQFSSET to provide input from MQFSSET file (that has already been loaded by the MQJSETUP.Z batch job):

```
// DLBL CONFIG,'WMQZVSE.mqfcnfg',,VSAM,CAT=?cat-name?
// DLBL MQFSSET,'WMQZVSE.mqfsset',,VSAM,CAT=?cat-name?
// LIBDEF PHASE,SEARCH=(PRD2.WMQZVSE,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPUTIL,SIZE=MQPUTIL
UPDATE FROM MQFSSET
/*
```

Note: The UPDATE command must have exclusive update access to the configuration file (MQFCNFG). The WebSphere MQ for z/VSE system using the configuration file cannot be active and the file cannot be open in CICS).

MQPEXCIC program

The MQPEXCIC utility can be used to alter VSE-specific MQ object attributes that cannot be altered using Programmable Command Formats (PCF) or WebSphere MQ commands (MQSC).

Specifically, the MQPEXCIC utility can be used to alter:
- Channel enabled flag.
- Channel dead-letter store flag.

The MQPEXCIC utility program uses the CICS Transaction Server EXCI facility and so is only available when WebSphere MQ for z/VSE runs in a CICS TS system. In addition, the utility requires:

- CICS TS GROUP DFH$EXCI to be installed.
- CICS TS IRC specified in the CICS startup JCL. IRC must be open in CICS.
- If CICS TS is running with SEC=YES, then the batch job must specify a // ID statement with a valid user ID and password.

MQPEXCIC accepts input from SYSIPT. The syntax for this input is:

```
CHANNEL(channel-name)  optional-parameters
```

where *channel-name* should match the name of a channel defined to the queue manager, and *optional-parameters* must include one or more of the following:

DLQSTORE (Y/N)
Dead Letter Store flag

ENABLE (Y/N)
Enable flag

The following example shows how the MQPEXCIC utility can be run as a batch job:

```
// JOB MQJEXCIC
// ID USER=userid,PWD=userpwd
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPEXCIC,PARM='vtam-applid',OS390
CHANNEL(my.channel) DLQSTORE(Y)  ENABLE(Y)
/*
/&
```

In this example, *vtam-applid* is the VTAM APPLID of the CICS/TS system running the WebSphere MQ for z/VSE queue manager. An optional keyword ",TRACE" may be added to the *vtam-applid* parameter to include diagnosis information in the batch output log.

Note: The MQPEXCIC utility requires OS390 emulation.

Using the batch interface

Unlike WebSphere MQ for other platforms, WebSphere MQ for z/VSE is implemented as a CICS subsystem. This means that access to WebSphere MQ objects using the message queue interface (MQI) is restricted to CICS applications. To avoid this limitation, WebSphere MQ for z/VSE provides an interface for batch programs.

The batch interface is designed to standardize the programming style of CICS and batch programs. From a programming point of view, batch programs use calls exactly the same way as CICS programs, that is, WebSphere MQ batch programs issue calls such as MQCONN, MQOPEN, and MQPUT to access WebSphere MQ objects.

Because WebSphere MQ objects are ultimately under the control of the CICS subsystem, calls issued by batch programs are passed to the CICS partition for processing. This is achieved using cross partition communication calls (XPCC). Batch programs are not concerned with XPCC, because all relevant logic is built into MQI calls.

WebSphere MQ for z/VSE provides a special CICS transaction, MQBI, that must be running to process MQI calls issued by batch programs. This transaction must be running for the batch interface to be available. MQBI waits for MQCONN calls issued by batch programs. When these are received, MQBI starts a second transaction, MQBX, which issues all MQI calls on behalf of the batch program. There is one MQBX instance for each active batch connection.

The MQBX transaction runs for the duration of the logical WebSphere MQ connection, that is, it runs until the batch program issues an MQDISC. If a batch program issues a second MQCONN call, the batch interface starts a second MQBX transaction for the duration of that WebSphere MQ connection. This design allows batch programs to create logical units of work. It also means that multiple batch programs (including multiple z/VSE subtasks) can establish concurrent connections to the WebSphere MQ queue manager.

Note: Using the batch interface adds a performance overhead, because MQI calls issued from batch programs are transferred to mirror CICS transactions.

Each WebSphere MQ queue manager running on a z/VSE system can activate a single batch interface. Each interface is identified by a unique batch identifier name. Batch identifiers must be unique within the context of the z/VSE system (that is, remote z/VSE systems can use the same identifier names, but queue managers running on the same z/VSE system must be configured with unique identifiers).

The queue manager's batch identifier is configured in the global system definition as a communications parameter (press PF9 from MQMT option 1.1):

```
2011/10/31        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
21:29:46                  Global System Definition                   CIC1
MQWMSYS                    Communications Settings                   A000

    TCP/IP settings                          Batch Interface settings
    Licensed clients . . : 00000             Batch Int. identifier: MQBSRV39
    Adopt MCA  . . . . . : N                 Batch Int. auto-start: Y
    Adopt MCA Check  . . : N
                                             Channel Auto-Definition
                                             Auto-definition  . . : N
    SSL parameters                           Auto-definition exit :
    Key-ring sublibrary  :
    Key-ring member  . . :
    SSL reset count  . . :

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : Y

Requested record displayed.
PF2=Return PF3=Quit PF4/Enter=Read PF6=Update PF10=Listeners PF11=Services
```

Figure 70. Batch interface identifier

The batch interface settings, as part of the communication parameters of the queue manager's global system definition, include the batch interface identifier and the batch interface auto-start parameters.

Batch interface identifier

The batch interface identifier is an 8-character alphanumeric name that uniquely identifies the queue manager to batch MQI programs.

Batch programs identify which queue manager to connect to by including a `// SETPARM` card in their JCL. The SETPARM parameter required to identify a queue manager is MQBISRV. For example:

```
// SETPARM MQBISRV='MQBISRV2'
```

Since batch JCL can only specify one SETPARM for the MQBISRV symbol, batch programs can only connect to one queue manager per submission, even if there are multiple queue managers running on the z/VSE system.

The default value for the batch interface identifier is MQBISERV. This is also the default for batch programs that do not supply a //SETPARM card in their JCL.

If two queue manager's running on the same z/VSE system rely on the identifier default, or are configured with the same identifier name, the queue manager that services batch programs is unpredictable.

Batch interface auto-start

The batch interface auto-start parameter indicates whether or not the queue manager should automatically start the batch interface (transaction MQBI) during system initialization.

The auto-start parameter can be set as follows:

Y Start the batch interface during queue manager initialization.

N Do not start the batch interface during queue manager initialization.

By choosing to start the batch interface during queue manager initialization, the batch interface will also be stopped automatically during queue manager shutdown.

Starting the batch interface

The batch interface is started by running the MQBI transaction. This can be done in native CICS, or by configuring CICS to run the batch interface start program (MQPSTBI) during post initialization. MQBI is a long-running CICS transaction that coordinates multiple simultaneous batch connections to the WebSphere MQ queue manager.

Alternatively, the batch interface can be started automatically during queue manager initialization by switching on the batch interface auto-start parameter in the communications settings of the global system definition.

Stopping the batch interface

The batch interface can be stopped normally in one of three ways:

- In native CICS.
- From a batch program.
- During CICS system shutdown.

The batch interface can be stopped abnormally by shutting down CICS with the immediate option, or by purging or forcing the MQBI transaction. The MQBISTOP sample program provides an example of stopping the interface from a batch program.

The batch interface can be stopped during CICS system shutdown by configuring the CICS PLTSD to run program MQPSPBI. If so configured, the PLT macro for MQPSPBI should precede the MQ shutdown program MQPSTOP. For example:

```
DFHPLT TYPE=ENTRY,PROGRAM=MQPSPBI
DFHPLT TYPE=ENTRY,PROGRAM=MQPSTOP
```

The batch interface is stopped automatically if the batch interface auto-start parameter is set on in the communications settings of the global system definition.

How to use the batch interface

1. Issue WebSphere MQ functions in your batch program, just as you do in CICS programs. For example:

```
CALL 'MQCONN'  USING
     QM-NAME-AREA
     HCONN-ADDR-AREA
     CCODE-ADDR-AREA
     RCODE-ADDR-AREA.
```

2. Link-edit your program by including module MQBIBTCH. For example:

```
// JOB MQBATBLD
// OPTION CATAL
PHASE MYPROG,*
// EXEC IGYCRCTL,....
    your program here
/*
INCLUDE MQBIBTCH
// EXEC LNKEDT
/&
```

3. Start the CICS interface, using the transaction MQBI.
4. Run your batch program.

   The following JCL might be used to run a WebSphere MQ batch application called MYPROG:

```
// JOB MQBATRUN
// SETPARM MQBISRV='MQBISRV2'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// EXEC MYPROG
/*
/&
```

   Note that the SETPARM card identifies the queue manager the batch program will connect to, and the LIBDEF SEARCH path must include the WebSphere MQ installation library. If you omit the SETPARM for the MQBISRV symbol, the batch program will attempt to communicate with a queue manager configured with the batch identifier name MQBISERV.

Data integrity

To test for data integrity the following functions are used:

- MQCMIT commits all changes. This forces a CICS SYNCPOINT to be issued by the mirror transaction.
- MQBACK rolls back all changes. The CICS mirror transaction issues `EXEC CICS SYNCPOINT ROLLBACK`.

For both functions the syntax is as follows:

```
CALL 'funct' USING
    HCONN-ADDR-AREA
    CCODE-ADDR-AREA
    RCODE-ADDR-AREA.
```

Note:

1. None of the passed parameters is actually tested or used.
2. Under CICS, updates are not automatically committed. However, if a batch program issues the MQDISC call while there are uncommitted requests, an implicit syncpoint occurs.

Verifying the batch interface

The batch program MQBICALL has been provided for this purpose. You can use the following job as a test:

```
// JOB CALLER
// SETPARM MQBISRV='batchid'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
*  Put 5 messages into queue: ANYQ
// EXEC MQBICALL
PUT  05 ANYQ
/*
/&
```

The MQBICALL utility is provided with the WebSphere MQ installation library as both an executable and a COBOL source file. It can be used as a basis for your own WebSphere MQ batch applications.

Note that the SETPARM for symbol MQBISRV should be changed to identify an appropriate queue manager batch interface identifier.

Restrictions on using the batch interface

1. Message lengths are restricted to 250k.
2. The MQINQ and MQSET MQI calls are limited as follows:
   - A maximum of 10 selectors.
   - A maximum of 10 integer attributes.
   - 500 characters for character attribute buffer.
3. When the batch interface is started, a GETVIS for approx 260K of above the line storage is issued. This GETVIS storage is used to support the next batch job to connect to this batch interface. When a batch job starts, the batch interface will start an interface task (MQBX) and pass it this previously acquired storage. The MQBX task will use this storage for passing data to and from the batch job. The batch interface will then acquire another work area (a GETVIS of 260K of above the line storage) which will be used by the next batch job to connect to the batch interface.

   If this storage is not available then an MQBI0110W console message will be issued to warn that any batch job attempting to connect will fail with CC=0 RC=2059. Only when a currently executing batch job finishes will the batch interface be able to receive a connection from another batch job.

   The batch interface will terminate with MQBI0101E console message if it is unable to obtain the initial 260K of above the line storage.

Batch interface and the client bridge

The WebSphere MQ for z/VSE Batch Interface works co-operatively with the WebSphere MQ client for z/VSE bridge. Batch applications, link-edited with the Batch Interface MQI will run client connections if they identify a client bridge by means of the // SETPARM MQBISRV.

Similarly, client applications, link-edited with the client bridge MQI will run batch connections if they identify a Batch Interface via the // SETPARM MQBISRV.

The client bridge is described in “The WebSphere MQ client bridge” on page 637.

VSAM file maintenance

All files used by WebSphere MQ are VSAM clusters. Most of these contain queues and need to be reorganized from time to time.

A queue is an ordered suite of VSAM records in a KSDS organization. Each record key is 56 bytes long, 48 being for the queue name, and eight for the Queue Sequence Number (QSN) and other information. This QSN is assigned sequentially, resulting in all keys being created in ascending order.

Even when a queue record is physically deleted from a queue, the space it occupies is not reclaimed due to the way VSAM works. Therefore, unless you reclaim the space used by these records, there is the possibility that you will obtain a VSAM "space full" condition.

The queue dump facility allows you to rebuild an WebSphere MQ VSAM queue file. This eliminates processed messages and fully regains VSAM freespace.

There are three ways to reclaim the space of deleted messages :

1. Use the MQPREORG utility.
2. Perform a VSAM DELETE and DEFINE to recreate the VSAM dataset. Only do this if your queue is empty. If you have multiple queues in a single VSAM file (not recommended), all queues should be empty.
3. Use the automatic reorganization feature available with your queue definition. Automatic reorganization is available only for single queues defined in a single VSAM file.

Delete all function

On the Maintain Queue Records screen (see "Queue maintenance" on page 142), there is a function called "Delete All". This function physically deletes all messages, and resets the QSN to one, in order to reclaim freed space.

This is a useful tool to maintain the system log file for WebSphere MQ. The advantage of this function is that it is an online function requiring no other manual operation.

Attention: Note that this function deletes all messages and should not be used on queue files that contain undelivered messages. It is not recommended that the DELETE ALL option be used to maintain production queues as this can leave VSAM indexes fragmented and lead to poor performance. Instead, the automatic reorganization feature or the batch MQPREORG utility should be considered.

Operation

1. On the Start/Stop Queue Control screen, stop the desired queue; see Figure 54 on page 136.
2. If the desired queue is a transmission queue, stop only the inbound direction first. When the queue depth reaches zero, stop the outbound direction and close the associated sender channel.
3. If the desired queue is a destination queue with trigger capability, close the associated receiver channel.
4. On the Maintain Queue Records screen enter the queue name, together with a function of A, and press the PF6 (Update) function key; see Figure 58 on page 143.
5. Press the Enter key to display the result.

6. After "Queue Processing Finished" is displayed, start the reorganized queue on the Start/Stop Queue control screen.

MQPREORG function

WebSphere MQ includes a batch program utility called MQPREORG, and sample JCL to run MQPREORG.

This utility can be used as a nightly, or weekly, queue maintenance facility on any number of queue files. You can also specify a date and time to carry out the procedure. The utility accepts the queue name from SYSIPT and the name of the VSAM file from DLBL.

All messages are ignored, except those marked as "Written" (to be delivered after a specified date and time) on the specified queue. The retained messages are resequenced and placed in a workfile.

After the VSAM cluster is deleted and redefined, the retained and resequenced messages are copied back into it. If none of the written messages is to be retained, you can use a "delete-and-define" IDCAMS JCL to do the job.

Multiple queues sharing a VSAM cluster

Attention: Although it is possible for WebSphere MQ for z/VSE queues to share the same VSAM file cluster, this is **not** advised. To give maximum independence to data, each queue should be assigned a unique VSAM file cluster.

This is particularly important if the queue is defined for automatic reorganization. See "Creating local queues" on page 97.

If there is more than one queue defined in a VSAM cluster, all queues have to be processed before deleting and recreating this cluster. Otherwise, records from unprocessed queues will be lost.

To help you reorganize all queues, you may use the "ALL" option instead of the queue name, as follows:

```
// EXEC MQPREORG
ALL
/*
```

To reorganize a specific queue, enter one of the following commands:

```
// EXEC MQPREORG
LQ.INVOICE
/*
```

or

```
// EXEC MQPREORG
LQ.INVOICE YYYYNNDDHHMMSS
/*
```

where:

YYYY Is the year
NN Is the month
DD Is the day
HH Is the hour
MM Is the minute
SS Is the second

Reorganizing queue files

This procedure is to be used only when the queue manager is not running.

1. If CICS is running, use CEMT to shut down and close the VSAM files you are going to process.
2. Modify the sample JCL to include your system parameters and reorganization requirements.
3. Process the job to run the batch program utility, MQPREORG, to reorganize the VSAM files and reclaim all freed space.
4. If you performed step 1, use CEMT to open and enable the processed VSAM files.

Sample JCL to run MQPREORG

```
* ** JOB JNM=MQJREORG,DISP=D,CLASS=0
* ** LST DISP=H,CLASS=Q,PRI=3
// JOB MQJREORG - Re-Organize WebSphere MQ for z/VSE queues.
* ------------------------------------------------------------------*
*      I M P O R T A N T    I M P O R T A N T  I M P O R T A N T    *
*                                                                   *
*   Please change :                                                 *
*                 "*  ** JOB" to  "* $$ JOB"                        *
*                 "*  ** LST" to  "* $$ LST"                        *
*                 "*  ** EOJ" to  "* $$ EOJ"                        *
*                                                                   *
*   Fields filled with ?volid? have also to be modified to suit the *
*  user specifications.                                             *
*                                                                   *
* ------------------------------------------------------------------*
*                                                                   *
*   This job deletes delivered messages from an WebSphere MQ queue  *
*   in order to reclaim the DASD freed space.                       *
*                                                                   *
*   INPUT to MQPREORG :                                             *
*   (only one statement is allowed, delimited by one or more spaces)*
*                                                                   *
*     1. Any QUEUE name delimited by one or more spaces             *
*        (In this JCL, only queue OS2_LOCAL is to be processed)     *
*        If there are any other queues in the same cluster,         *
*        they will be echoed into OUTPUTQ.                          *
*     2. If you want to process EVERY queue in a cluster,           *
*        please key in "ALL ".                                      *
*                                                                   *
*   This sample assumes we want to reorganize queues defined to the *
*   VSAM cluster MQFI002. Changes must be made for other clusters.  *
* ------------------------------------------------------------------*
* Licensed Materials - Property of IBM                              *
*                                                                   *
* 5655-U97                                                          *
* Copyright IBM Corp. 2008                                          *
*                                                                   *
* US Government Users Restricted Rights - Use, duplication or       *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
* ------------------------------------------------------------------*
// DLBL INPUTQ,'WMQZVSE.MQFI002',,VSAM,CAT=MQMCAT
// DLBL OUTPUTQ,'WMQZVSE.WORK.QUEUE',,VSAM,CAT=MQMCAT
// EXEC IDCAMS,SIZE=AUTO
 /*                                                                */
 /*                  VERIFY VSAM FILE                              */
 /*                                                                */
          VERIFY FILE(INPUTQ)
          IF MAXCC > 0 THEN CANCEL   /* This means Cluster in use  */

          DELETE   (WMQZVSE.WORK.QUEUE)                    -
```

```
                  CL ERASE PURGE CAT(?CAT?)
          SET MAXCC = 0
          DEFINE CLUSTER                                      -
                   (NAME (WMQZVSE.WORK.QUEUE)                -
                    CYLINDERS (10 10)                         -
                    VOLUMES (?volid?)                         -
                    NONINDEXED)                               -
                 DATA                                         -
                   (NAME (WMQZVSE.WORK.QUEUE.DATA)           -
                    RECORDSIZE (57   32703)                   -
                    CISZ (8096))                              -
                    CAT (?CAT?)
/*
// IF $MRC GT 0 THEN
// GOTO WRAPUP
// LIBDEF PHASE,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// EXEC MQPREORG,SIZE=AUTO
OS2_LOCAL
/*
// IF $MRC GT 0 THEN
// GOTO WRAPUP
// EXEC IDCAMS,SIZE=AUTO
          DELETE   (WMQZVSE.MQFI002)                          -
                    CLUSTER NOERASE PURGE CATALOG (?CAT?)
     SET MAXCC = 0
 /*                                                           */
          DEF CLUSTER(NAME(WMQZVSE.MQFI002)                   -
              FILE(MQFI002)                                    -
              VOL(?volid?)                                     -
              RECORDS (3000 100)                               -
              RECORDSIZE (200 4089)                            -
              INDEXED                                          -
              KEYS(56 0)                                       -
              SHR(2))                                          -
              DATA (NAME (WMQZVSE.MQFI002.DATA) CISZ(4096))    -
              INDEX (NAME (WMQZVSE.MQFI002.INDEX) CISZ(1024))  -
              CATALOG(?CAT?)
          IF LASTCC > 0 THEN  CANCEL
  /*                                                          */
  /*         Execute REPRO only of the define  was OK.        */
  /*                                                          */
         REPRO INFILE(OUTPUTQ) OUTFILE(INPUTQ)
         IF LASTCC > 0 THEN  CANCEL
  /*                                                          */
  /*             Delete only if REPRO was OK.                 */
  /*                                                          */
         DELETE   (WMQZVSE.WORK.QUEUE)                   -
                    CL ERASE PURGE CAT(?CAT?)
/*
/. WRAPUP
/&
* ** EOJ
```

WebSphere MQ-CICS Bridge

The WebSphere MQ-CICS bridge enables an application, not running in a CICS environment, to run a program or transaction on CICS and get a response back. This non-CICS application can be run from any environment that has access to an WebSphere MQ network that encompasses WebSphere MQ for z/VSE.

A *program* is a CICS program that can be invoked using the EXEC CICS LINK command. It must conform to the DPL subset of the CICS API. That is, it must not use CICS terminal or syncpoint facilities.

A *transaction* is a CICS transaction designed to run on a 3270 terminal. The transaction can use BMS or TC commands. It can be conversational or part of a pseudoconversation. It is permitted to issue syncpoints. For further details about the transactions that can be run, see Part 5 of the *CICS Internet and External Interfaces Guide* ("Bridging to 3270 transactions").

When to use the CICS bridge

The CICS bridge allows an application to run a single CICS program or a "set" of CICS programs (often referred to as a unit of work). It caters for the application that waits for a response to come back before it runs the next CICS program (synchronous processing), and for the application that requests one or more CICS programs to run but does not wait for a response (asynchronous processing).

The CICS bridge also allows an application to run a 3270-based CICS transaction, without knowledge of the 3270 data stream.

The CICS bridge uses standard CICS and WebSphere MQ security features and can be configured to authenticate, trust, or ignore the requestor's user ID.

Given this flexibility, there are any many instances where the CICS bridge can be used. For example, when you want to:

- Write a new WebSphere MQ application that needs access to logic or data (or both) that reside on your CICS server.
- Be able to run CICS programs from a Lotus® application.
- Be able to access your CICS applications from:
  - Your WebSphere MQ Classes for Java client application.
  - A web browser using the WebSphere MQ Internet gateway.

For information about how to write an WebSphere MQ-CICS bridge application, see the *WebSphere MQ Application Programming Guide*, or the *Using WebSphere MQ for z/VSE* redbooks publication, SG24-5647-01.

System configuration for the CICS bridge

When you are setting your system up, you should ensure that:

- Both WebSphere MQ and CICS are running in the same z/VSE system.
- The WebSphere MQ request queue is local to the CICS bridge. The response queue, however, can be local or remote.
- The CICS bridge tasks run in the same CICS as the bridge monitor. The user programs can be in the same or a different CICS system.

Running CICS DPL programs

Data necessary to run the program is provided in the WebSphere MQ message. The bridge builds a COMMAREA from this data, and runs the program using EXEC CICS LINK.

The following takes place when running CICS DPL programs over the MQ CICS Bridge:

1. A message, with a request to run a CICS program, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a "start unit of work" message is waiting (CorrelId=MQCI_NEW_SESSION).

3. Relevant authentication checks are made, and a CICS DPL bridge task is started with the appropriate authority (see "Security considerations for the CICS bridge" on page 192).
4. The CICS DPL bridge task removes the message from the request queue.
5. The CICS DPL bridge task builds a COMMAREA from the data in the message and issues an EXEC CICS LINK for the program requested in the message.
6. The program returns the response in the COMMAREA used by the request.
7. The CICS DPL bridge task reads the COMMAREA, creates a message, and puts it on the reply-to queue specified in the request message. All response messages (normal and error, requests and replies) are put to the reply-to queue with default context.
8. The CICS DPL bridge task ends.

A unit of work can be just a single user program, or it can be multiple user programs. There is no limit to the number of messages you can send to make up a unit of work.

In this scenario, a unit of work made up of many messages works in the same way, with the exception that the CICS bridge task waits for the next request message in the final step unless it is the last message in the unit of work.

Running CICS 3270 transactions

Data necessary to run the transaction is provided in the WebSphere MQ message. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more MQ messages to communicate between the CICS transaction and the WebSphere MQ application.

Unlike traditional 3270 emulators, the bridge does not work by replacing the VTAM flows with WebSphere MQ messages. Instead, the message consists of a number of parts called vectors, each of which corresponds to an EXEC CICS request. The application is therefore talking directly to the CICS transaction, rather than by means of an emulator, using the actual data used by the transaction (known as *application data structures* or ADSs).

The following takes place when running CICS 3270 transactions over the MQ CICS Bridge:

1. A message, with a request to run a CICS transaction, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a "start unit of work" message is waiting (CorrelId=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS 3270 bridge task is started with the appropriate authority (see "Security considerations for the CICS bridge" on page 192).
4. The MQ-CICS bridge exit removes the message from the queue and changes task to run a user transaction.
5. Vectors in the message provide data to answer all terminal-related input EXEC CICS requests in the transaction.
6. Terminal-related output EXEC CICS requests result in output vectors being built.
7. The MQ-CICS bridge exit builds all the output vectors into a single message and puts this on the reply-to queue.

8. The CICS 3270 bridge task ends.

Note: The CICS bridge exit is an WebSphere MQ-supplied CICS exit associated with the bridge transaction.

A traditional CICS application usually consists of one or more transactions linked together as a pseudoconversation. In general, each transaction is started by the 3270 terminal user entering data onto the screen and pressing an AID key. This model of application can be emulated by an WebSphere MQ application. A message is built for the first transaction, containing information about the transaction, and input vectors. This is put on the queue. The reply message consists of the output vectors, the name of the next transaction to be run, and a token that is used to represent the pseudoconversation. The WebSphere MQ application builds a new input message, with the transaction name set to the next transaction and the facility token set to the value returned on the previous message. Vectors for this second transaction are added to the message and the message put on the queue. This process is continued until the application ends.

An alternative approach to writing CICS applications is the conversational model. In this model, the original message might not contain all the data to run the transaction. If the transaction issues a request that cannot be answered by any of the vectors in the message, a message is put onto the reply-to queue requesting more data. The WebSphere MQ application receives this message and puts a new message back to the queue with a vector to satisfy the request.

For more information, see the *CICS Internet and External Interfaces Guide*.

Customizing the CICS bridge

Before you can run the bridge, you must ensure that your VSE system has both the CICS and WebSphere MQ components in place:

1. On your CICS system:

   Run the resource definition utility DFHCSDUP, using the sample MQJCSD24.Z as input, to define the bridge transactions and programs. Note that this sample contains definitions for all WebSphere MQ for z/VSE CICS objects. The MQJCSD24.Z JCL sample is normally run when WebSphere MQ is installed, and may not need to be rerun.

   MQ Bridge-related definitions include:

   CKBR Bridge monitor transaction

   CKBP Bridge ProgramLink transaction

   CSQCBR00
   Bridge monitor program

   CSQCBP00
   Bridge ProgramLink program

   CSQCBP10
   Bridge ProgramLink abend handler program

   CSQCBE30
   3270 bridge exit for WebSphere MQ

   CSQCBDCI
   Bridge data conversion program for requests

   CSQCBDCO
   Bridge data conversion program for replies

Note:

a. The bridge requires CICS Transaction Server for z/VSE Version 1 Release 1.1, or later.

b. The bridge uses CICS temporary storage IDs with the prefix CKB. You should make sure these are not recoverable.

c. By default, your CICS DPL programs run under transaction code CKBP. You need to change the TASKDATALOC attribute to "BELOW" if you are going to run 24-bit programs, otherwise you will get a CICS abend AEZC. If you want to run your programs under different transaction codes, you need to install copies of the definition of CKBP, changing the transaction names to ones of your choice. DPL bridge transactions must not be routed to a remote system.

d. Support for the MQ CICS Bridge was introduced to WebSphere MQ for z/VSE with APAR PQ93406.

2. On your WebSphere MQ system:

   a. Define a local queue for the request messages.

      The default MQ CICS Bridge request queue is SYSTEM.CICS.BRIDGE.QUEUE. You must define a queue specifically for bridge request messages. You can use the default name, or choose a different name. The request queue must have the following attribute:

      SHARE
      Allows both the monitor and the bridge tasks to read the queue.

      Note: The WebSphere MQ queue defined to hold requests for the CICS bridge must not be used by any other application (other than those putting requests to the queue). Each CICS bridge monitor task started requires its own WebSphere MQ queue to hold requests.

   b. Define one or more queues to hold the responses, as required. If your response queue is remote, you must define a transmission queue to hold the responses before they are forwarded to the response queue, and a remote queue that points to the transmission queue.

      If the bridge is to be accessed remotely from WebSphere MQ for z/VSE, you need channel and transmission queue definitions, and a remote queue definition for the request queue. For more information about using remote queues, see the *WebSphere MQ Intercommunication* manual.

3. Security

   You might need to add ESM definitions, depending on the authentication option you choose when starting the MQ CICS Bridge monitor. For more information about WebSphere MQ for z/VSE security and ESM definitions, see Chapter 12, "Security," on page 651.

Starting the CICS bridge

To start the bridge, you need to run the CKBR transaction with up to three parameters:

Q=*queue_name*
Where *queue_name* is the name of the queue holding requests.

If you do not specify a name, the default is:

```
SYSTEM.CICS.BRIDGE.QUEUE
```

Remember that names of objects within WebSphere MQ are case-sensitive.

AUTH=*sec_option*

Where *sec_option* is the security option. Valid security options include:

LOCAL
Bridge runs under CICS DFLTUSER. No userid or password checking is performed. This is the default.

IDENTIFY
Bridge runs under MQMD userid. Userid is trusted, no password checking is performed.

VERIFY_UOW
Bridge runs under MQMD userid if password in MQCIH of start UOW is valid.

VERIFY_ALL
As for VERIFY_UOW, password required and checked for all subsequent messages in UOW.

WAIT=*secs*

Where *secs* is the number of seconds that you want the bridge task to wait for second and subsequent requests before timing out when processing a unit of work that runs many user programs.

The default wait time is unlimited.

It is preferable to specify a wait time. If you do not specify a wait time, the CICS bridge might inhibit CICS or WebSphere MQ shut down.

DISCINT=*secs*

Where secs is the number of seconds that you want the bridge task to wait for a CICS bridge request message before terminating.

This option should be used when the bridge is started as a trigger, rather than as a long running transaction. If the bridge (transaction CKBR) is to be started as a trigger, the startup parameters are specified in the queue's USERDATA field, and it is recommended to use trigger type EVERY and MAX STARTS=1.

Start the CKBR task running by using one of the following methods:

- Input a single line from a terminal (3270 or other). Note that the terminal is not freed until the monitor ends. The format is:

  ```
  CKBR Q=queue_name,AUTH=sec_option,WAIT=secs
  ```

  For example:

  ```
  CKBR Q=MyQueue,AUTH=IDENTIFY,WAIT=30
  ```

- Issue an EXEC CICS START for the CKBR program with the parameters as data.
- Issue an EXEC CICS LINK to the program CSQCBR00 with the parameters as data in the commarea.
- In the local CICS bridge request queue specify TRIGTRAN('CKBR'), TRIGTYPE(EVERY) and MAXSTART(1), with any parameters for the AUTH, WAIT and DISCINT options in USERDATA.

If a high volume of requests is expected, you could consider starting a second or subsequent monitor task. To do this, you must create another request queue for the sole use of this monitor (and the bridge tasks it starts).

Shutting down the CICS bridge

You can shut down the CICS bridge by:

- Shutting CICS down.
- Shutting WebSphere MQ down.
- Using DISCINT when starting the Bridge. If no CICS bridge request message arrives on the request queue within the DISCINT number of seconds, then the bridge task will finish.

Whichever method you choose, it attempts to allow all the requests in progress to complete first. However, if this is not possible, the problems encountered are reported on the MQ system log.

Restarting the monitor

The monitor requires exclusive use of the request queue during its initialization. Consequently, the monitor cannot be restarted until all bridge tasks for the queue have terminated.

Security considerations for the CICS bridge

When you run the CICS bridge, you can specify the level of authentication you want to take place. If requested, the bridge checks the user ID and password extracted from the WebSphere MQ request message before running the CICS program named in the request message.

Note:

1. To fully exploit security in WebSphere MQ for z/VSE, you must have an External Security Manager (ESM) installed and enabled, and the MQ security feature also enabled.
2. If you have not specified a user ID or password in a message, the bridge task runs with the LOCAL level of authentication, even if you started the bridge monitor with a different authentication option.
3. The options that include password (or passticket) validation require a CICS bridge header (MQCIH) to be provided. For more information about the MQCIH header, see the *WebSphere MQ Application Programming Reference*.

The level of authentication you can use is:

LOCAL
This is the default. CICS programs run by the bridge task are started with the CICS DFLTUSER user ID and therefore run with the authority associated with this user ID. There is no checking of user IDs or passwords. If a CICS program is run that tries to access protected resources, it will probably fail.

IDENTIFY
When you start the monitor task with the IDENTIFY authentication option, the bridge task is started with the user ID specified in the message (MQMD). CICS programs run by the bridge run with the user ID extracted from the MQMD. There is no password checking and the user ID is treated as trusted.

VERIFY_UOW
When you start the monitor task with the VERIFY_UOW authentication option, the monitor task checks the user ID and password by issuing the EXEC CICS VERIFY PASSWORD command before starting the bridge task.

CICS programs run by the bridge run with the user ID extracted from the MQMD. If the user ID or password is invalid, the request fails with return code MQCRC_SECURITY_ERROR.

VERIFY_ALL
This is the same as VERIFY_UOW except that the bridge task checks the user ID and password in every message. This is not applicable for 3270 transactions.

If you have not specified a user ID in a message, or you have not provided a password, the CICS program started by the CICS bridge runs with the user ID set to the CICS DFLTUSER, regardless of the option requested. If you want more than one level of authentication checking performed, run a monitor task for each level you need.

Using and writing WebSphere MQ-CICS bridge applications

The WebSphere MQ-CICS bridge was originally available for WebSphere MQ for OS/390® only.

Extensive information regarding the bridge and, in particular, how to use and write bridge applications, can be found in the *WebSphere MQ Application Programming Guide* and the *WebSphere MQ Application Programming Reference*.

In addition, information about the CICS Bridge in general can be found in the *CICS Transaction Server for z/VSE Version 1 Release 1.1* and the *CICS External Interfaces Guide*.

Chapter 6. Problem determination

This chapter suggests reasons for some of the problems you may have using WebSphere MQ for z/VSE. The process of problem determination is that you start with the symptoms and trace them back to their cause.

Not all problems can be solved immediately, for example, performance problems caused by the limitations of your hardware. Also, if you think that the cause of the problem is in the WebSphere MQ code, contact your IBM Support Center.

The cause of your problem could be in:
- WebSphere MQ setup and local queue operation.
- The network.
- The application.
- Other areas of investigation.

The sections that follow raise some fundamental questions that you need to consider. Work through the questions, making a note of anything that might be relevant to the problem.

WebSphere MQ setup and local queue operation

You should ensure that WebSphere MQ for z/VSE is installed correctly and working with local queues before you investigate any other problems.

Has WebSphere MQ run successfully before?

If WebSphere MQ has not run successfully before, it is likely that you have not yet set it up correctly. See “WebSphere MQ installation verification test” on page 32 to check that you have carried out all the steps correctly, and set up a SYSTEM.LOG queue as follows:

1. Define a queue name as SYSTEM.LOG using:
   a. A physical file name MQFLOG using the file name from the file control table.
   b. A maximum queue depth of 5 000.

   See step 4d on page 31 and step 4f on page 31 in “WebSphere MQ installation verification test” on page 32 for information about defining a queue.
2. Ensure the system log queue name, usually SYSTEM.LOG, is named in the queue manager's global system definition (accessible using MQMT option 1.1).

You can browse the log queue by selecting 4, Browse Queue Records, on the Master Terminal Main Menu, as described in “Browse function” on page 152.

See “Global system definition” on page 82 for more information.

Is local queue operation working?

This may require the creation of a local queue definition as described in “WebSphere MQ initialization” on page 28. In addition, check that the VSAM files referenced in the queue definition are open and correctly enabled.

Use:

```
CEMT INQUIRE FILE (filename)
```

to ensure that the VSAM file associated with the queue is accessible.

Use the instructions described in "Local queue verification test" on page 32 to test a local queue. Ensure that you can:

- Put and get messages to the local queue, using the supplied test transaction TST2.
- Browse the queue correctly using the MQMT System Administration Browse function.

Network problems

Before WebSphere MQ for z/VSE can use an inbound or outbound channel connection to an SNA-connected WebSphere MQ platform, a connection must already be established between CICS for z/VSE and the remote platform.

The person responsible for the VTAM and CICS definitions in your enterprise should perform the following investigations.

Investigating SNA problems

If an attempt to start a channel fails, it may be the result of a session failure. If it is not possible to establish a session between CICS and the LU for the remote channel endpoint, either before or during the channel attempting to start, the connection fails.

Enter the following command if you suspect that a session failure is causing the problem:

```
D NET,ID=<remote lu name>,E
```

This gives details of the LU which should be in session with CICS, and also lists any sessions it currently has.

Note:

1. Look at the session limit for the LU. If it is shown as one for an independent LU, there is a problem with the SNA definitions.
2. See if <minor node name> is listed amongst the sessions. If it is, there is a session between the LU and CICS. This indicates that the problem may not be at the network level, or that there are insufficient sessions between the two LUs to support a new channel request.

   Enter the command again, to see whether for this session, the send and receive counts have changed, indicating the session is in use.

If the command returns "PARAMETER VALUE INVALID", this means that VTAM does not know the <remote lu name>. Either you entered the name incorrectly, or VTAM cannot locate it. Try defining the name again and attempt to start the channel.

If VTAM is able to display <remote lu name>, try the following command in CICS:

```
CEMT I CONN(<remote conn>)
```

This shows the status of the connection from CICS to the remote system. Next to the entry is an indication showing it to be INService or OUTservice and ACQuired or RELeased. The status needs to be Inservice and Acquired.

```
CEMT I MODE CONN(<remote conn>)
```

This command displays the status of the mode names associated with the connection. For connections supporting parallel sessions, there will be at least two mode names, SNASVCMG and <logmode 1>, showing the number of active sessions for each.

If the SNASVCMG group has no sessions active, the connection is in a RELeased state, rather than an ACQuired state.

These sessions are SNA services manager sessions, and not used by WebSphere MQ channels. However, at least one of the two needs to be active for the connection to be usable.

If the remote LU has been incorrectly defined, so that it has a session limit of one, it is possible that one SNASVSMG session is active, but that no other sessions can be established, including those required by the WebSphere MQ channel.

The <logmode 1> sessions may be used by WebSphere MQ channels.

For single session connections, one mode name, <logmode 2>, is shown with just one session in the group.

The WebSphere MQ channel must have been set up to use the logon mode <logmode 1>, or <logmode 2>, as appropriate.

Investigating TCP/IP problems

Is TCP/IP able dynamically to establish a session between nodes in the network? Use the following instruction to test a connection to a remote TCP/IP node:

```
[ping hostname]
```

If you are unable to “ping” the remote TCP/IP node successfully, inform your z/VSE systems programmer who installed TCP/IP.

Under z/VSE, the ping command can be entered from the console. For example:

```
msg f7
AR 0015 1I40I  READY
F7-0111 IPN300I Enter TCP/IP Command
111 ping 127.0.0.1
F7 0109 TCP910I  Client manager connected. Generated on 10/28/01 at 23.57
F7 0109 TCP915I  PING
F7 0109 TCP910I  PING Ready:
F7 0109 TCP915I  SET HOST= 127.0.0.1
F7 0109 TCP910I  127.000.000.001
F7 0109 TCP910I  PING Ready:
F7 0109 TCP915I  PING
F7 0109 TCP910I  PING 1 was successful, milliseconds: 00011.
F7 0109 TCP910I  PING 2 was successful, milliseconds: 00000.
F7 0109 TCP910I  PING 3 was successful, milliseconds: 00000.
F7 0109 TCP910I  PING 4 was successful, milliseconds: 00000.
F7 0109 TCP910I  PING 5 was successful, milliseconds: 00003.
F7 0109 TCP910I  PING Ready:
F7 0109 TCP915I  QUIT
F7 0111 IPN300I Enter TCP/IP Command
```

Note that the IP address or hostname used should match the IP address or hostname used in a sender channel definition. For example, if messages are to be

sent from z/VSE to a remote system, a sender channel will be defined that identifies that remote system by either IP address or hostname.

It should also be noted that some systems can deactivate or restrict ping requests. If this is the case, the TCP/IP administrator of the remote system should provide an alternative method to test connectivity between remote systems.

When messages are to be sent to z/VSE, the remote system should be able to ping the z/VSE system. Once again, if ping activity is restricted or disabled, the TCP/IP administrator should provide an alternative method to test connectivity between the two systems.

Investigating SSL problems

If secure sockets layer (SSL) services are configured for a sender, receiver or server-connection channel, that channel may terminate prematurely if an SSL error occurs. Such failures generally involve error messages written to the SYSTEM.LOG. Assuming logging is active for error and critical messages, the SYSTEM.LOG should always be checked in the event of a channel failure. Note that the severity of messages logged to the system log can be set from MQMT option 1.1.

A channel is SSL enabled if the SSL Cipher Specification parameter is specified in the channel definition. All other SSL parameters are ignored if this parameter is not set (that is, it is left blank).

SSL enabled channels require SSL services installed and active on both the local and remote systems. In addition, the SSL configuration parameters associated with the local channel definition, generally, must match the parameters associated with the remote channel definition. These should be checked in the event of an SSL failure. Generally, SSL channels may fail due to one of the following situations:

- The key-ring sublibrary name defined in the Global System Definition communications parameters does not identify a valid SSL key-ring sublibrary.
- The key-ring member name defined in the Global System Definition communications parameters does not identify a valid member set in the key-ring sublibrary for .PRVK and .CERT files.
- The SSL cipher specification of the channel definition is invalid or not supported by both the local and remote SSL subsystems.
- SSL client authentication of the channel definition is required, but the remote system did not provide an X.509 PKI certificate during SSL initial negotiation.
- The SSL peer attributes of the channel definition identify specific features expected of the remote system's certificate that do not match.
- The local and remote SSL subsystems are incompatible or running different version levels.

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 been 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?

Applications

The errors in the following list illustrate the most common causes of problems encountered while running WebSphere MQ programs. You should 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 may mean that MQI 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.

Are there any error messages?

WebSphere MQ uses the system log to capture messages concerning the operation of WebSphere MQ itself, the queue manager, and error data coming from the channels that are in use. Check the system log to see if any messages have been recorded that are associated with your problem.

See "System log" on page 209 for information about the contents of the system log.

Are there any return codes explaining the problem?

If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, refer to the *WebSphere MQ Application Programming Reference* manual for a description of that return code.

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.
- 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.

Have any changes been made since the last successful run?

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 may 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?

Has the application run successfully before?

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 causes a rarely used path in the program to be invoked.
- 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 which is causing the problem? For example, have the queues been defined with the same message length or priority?

If the application has not run successfully before

If your application has not yet run successfully, you need to examine it carefully to see if you can find any errors.

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 linker, if applicable, to see if any errors have been reported.

If your application fails to translate, compile, or link, it will also fail to run if you attempt to invoke it.

If the documentation shows that each of these steps was accomplished without error, you should 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 “Applications” on page 199 for some examples of common errors that cause problems with WebSphere MQ applications.

Using the WebSphere MQ API monitor

By selectively using the WebSphere MQ API monitor, you can:
- Track precisely which WebSphere MQ API issues an application.
- Establish which return codes are passed.

The WebSphere MQ API monitor is started and stopped using the MQMT system administration transaction option 2.1, which is used to toggle the API monitor on and off.

```
12/31/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
12:03:36               Start / Stop Queue                            CIC1
MQWMSS                                                               A003

                          System Information
          System Status    : SYSTEM IS ACTIVE
          Queue Status     : Queuing System is active.
          Channel Status   : Channel System is active.
          Monitor Status   : Monitor is not active.
                         Single Queue Request
          Queue Name       :
          Function         :         S=Start, X=Stop,  R=Refresh from Config
          Mode             :         I=Inbound, O=Outbound, B=Both

          INBOUND Status   :
          OUTBOUND Status  :

                          Queuing System Request
          Function         : M       S=Start, X=Stop, or M=Monitor


Please enter a Queue name.
Enter=Display  PF2=Return  PF3=Exit  PF6=Update
```

Figure 71. API monitor

Once the API monitor is started, the application to be tested can be processed, and the API monitor stopped.

Note: The API monitor should be started for limited periods only. It traces the processing of all running applications, and consequently makes heavy usage of system resources.

After the API monitor is toggled off, the SYSTEM.MONITOR queue can be browsed using the MQMT system administration browse facility. Each message in the queue represents the result of an WebSphere MQ API call.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:40:22                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.MONITOR
    QSN Number : 00000001       LR-        0, LW-       12, DD-MQFMON
                    Queue Data Record
Record Status : Written.        PUT date/time  : 20061120084101
Message Size  : 00000428        GET date/time  :
Offset ....+....|....+....|....+....|....+....|....+....|....+....|....+....|
 00000 MQER........            01              CONN    ...../......./........
 00070 ..
 00140
 00210
 00280
 00350
 00420



Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                       PF9=Hex       PF11=MD    PF12=Monitor
```

Figure 72. API monitor - browse

An MQCONN request is displayed in Figure 72. However, for more information, display the text in hexadecimal format, as shown in Figure 73.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:41:09                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.MONITOR
    QSN Number : 00000001       LR-        0, LW-       12, DD-MQFMON
                    Queue Data Record
Record Status : Written.        PUT date/time  : 20061120084101
Message Size  : 00000428        GET date/time  :
Offset ....+....|....+....|....+....|....+....|....+....|....+....|....+....|
 00000 MQER........            01              CONN    ...../......./........
       DDCD0000000344444444444FF4444444444444CDDD4444037006000370060000000 0
       485900000002000000000000010000000000000036550000033081 4C0330815C000000
 00070 ..
       00444444444444444444444444444444444444444444444444444444444444444444
       00000000000000000000000000000000000000000000000000000000000000000000
 00140
       44444444444444444444444444444444444444444444444444444444444444444444
       00000000000000000000000000000000000000000000000000000000000000000000

Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Char      PF11=MD    PF12=Monitor
```

Figure 73. API monitor - hexadecimal format

The layout of the displayed message is as follows:

```
01  MQ-MONITOR-RECORD.
   05  MQ-MON-TRAN-ID          PIC  X(4)  VALUE SPACES.
   05  MQ-MON-TERM-ID          PIC  X(4)  VALUE SPACES.
   05  MQ-MON-CICS-TASKN       PIC S9(8)  COMP VALUE ZERO.
   05  FILLER                  PIC  X(12) VALUE SPACES.
```

```
05  MQ-MON-SYSTEM-NUM       PIC  99    VALUE 01.
05  FILLER                  PIC  X(14) VALUE SPACES.
05  MQ-MON-FUNCTION         PIC  X(4)  VALUE SPACES.
   88 MQ-MON-CONNECT                VALUE 'CONN', 'CONI'
                                       'MCCO'.
   88 MQ-MON-CONNECT-VIA-APPL       VALUE 'CONN', 'CONI'.
   88 MQ-MON-CONNECT-VIA-INTERFACE VALUE 'CONI'.
   88 MQ-MON-MCP-CONNECT            VALUE 'MCCO'.
   88 MQ-MON-OPEN                   VALUE 'OPEN'.
   88 MQ-MON-PUT                    VALUE 'PUT '.
   88 MQ-MON-INQ                    VALUE 'INQ '.
   88 MQ-MON-GET                    VALUE 'GET '.
   88 MQ-MON-CLOSE                  VALUE 'CLOS'.
   88 MQ-MON-DISCONNECT             VALUE 'DISC'.

05  FILLER                  PIC  X(4)   VALUE SPACE.
05  MQ-MON-START-ABSTIME    PIC S9(15) COMP-3 VALUE ZERO.
05  MQ-MON-END-ABSTIME      PIC S9(15) COMP-3 VALUE ZERO.
05  MQ-MON-RESULTS.
    10  MQ-MON-CCODE        PIC S9(8)  COMP VALUE ZERO.
    10  MQ-MON-RCODE        PIC S9(8)  COMP VALUE ZERO.

05  FILLER                  PIC  X(12) VALUE SPACES.
05  MQ-MON-FUNCTION-DEP-INFO  VALUE SPACES.
    10  MQ-MON-QM-NAME      PIC  X(48).
    10  MQ-MON-Q-NAME       PIC  X(48).
    10  MQ-MON-RESOLVED-Q   PIC  X(48).
    10  FILLER              PIC  X(200).
```

PF12 activates a special monitor information screen as follows:

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:42:27                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.MONITOR
    QSN Number : 00000001       LR-        0, LW-       12, DD-MQFMON
                    Queue Data Record
Record Status : Written.        PUT date/time  : 20061120084101
Message Size  : 00000428        GET date/time  :
Queue line.
Transaction ID. . . MQER
Terminal ID . . . .
CICS Task . . . . .    50
Function. . . . . . CONN
Start Date/Time . . 2006/11/20  08:41:01
End   Date/Time . . 2006/11/20  08:41:01         0.01
Result  (CC RC) . . 0000 0000
QM. . . . . . . . .
Q . . . . . . . . .
Resolved Q. . . . .
Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                       PF9=Hex       PF11=MD    PF12=Queue
```

Figure 74. API monitor - monitor information

It is possible to follow the flow of WebSphere MQ API calls using a specific application. The application is identified by its CICS transaction code, terminal identifier, and CICS task number.

The specific WebSphere MQ API calls are identified, together with the queue manager name, queue name, and condition and return codes.

Ensure that you toggle the WebSphere MQ API monitor **off** after use.

Other areas of investigation

Perhaps the preliminary checks have enabled you to find the cause of the problem. If so, you should now be able to resolve it, possibly with the help of other books in the WebSphere MQ library (see "Bibliography" on page 1105) and in the libraries of other licensed programs.

If you have not yet found the cause, you must start to look at the problem in greater detail.

The purpose of this section is to help you identify the cause of your problem if the preliminary checks have not enabled you to find it.

When you have established that no changes have been made to your system, and that there are no problems with your application programs, choose the option that best describes the symptoms of your problem.

If none of these symptoms describe your problem, consider whether it might have been caused by another component of your system.

Have you obtained incorrect output?

In this book, "incorrect output" refers to your application:

- Not receiving a message that it was expecting.
- Receiving a message containing unexpected or corrupted information.
- Receiving a message that it was not expecting, for example, one that was destined for a different application.

In all cases, check that any queue or queue manager aliases that your applications are using are correctly specified and accommodate any changes that have been made to your network.

If an WebSphere MQ error message is generated, all of which are prefixed with the letters "MQI", you should look in the system log. See "System log" on page 209 for further information.

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it is dependent 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 failing to take into account the fact that processes can run independently of each other. For example, a program may issue an MQGET call, without specifying a wait option, before an earlier process has completed. An intermittent problem may also be seen if your application tries to get a message from a queue while the call that put the message is in-doubt (that is, before it has been committed or backed out).

Have you applied any service updates?

If a service update has been applied 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 had been applied correctly and completely?
- Does the problem still exist if WebSphere MQ is restored to the previous service level?
- If the installation was successful, check with the IBM Support Center for any patch error.
- If a patch has been applied to any other program, consider the effect it might have on the way WebSphere MQ interfaces with it.

Does the problem affect only remote queues?

If the problem affects only remote queues, check the following:

- Check that required channels have been started and are triggerable.
- 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.
- Check the system log and z/VSE console for messages indicating channel errors or problems.

See the *WebSphere MQ Intercommunication* book for information about how to define channels.

Is your application or WebSphere MQ for z/VSE running slowly?

WebSphere MQ for z/VSE runs as a subsystem, with a CICS partition, on the z/VSE operating system. The z/VSE operating system itself may be a second-level client on a VM machine. This complexity means that a performance problem can exist in any of these components.

WebSphere MQ for z/VSE is sensitive to the CICS environment and availability of CICS resources. CICS performance problems are a specialized area requiring detailed analysis. Investigate these problems with the assistance of your CICS systems programmer.

WebSphere MQ for z/VSE utilizes VSAM files under z/VSE. After prolonged use these files tend to fragment into several VSAM extents. This can be viewed with the VSE ICCF File and Catalog Management facility. Any files that show multiple extents should be reallocated with IDCAMS as soon as it is convenient to do so.

If your application is running slowly, this could indicate that it is in a loop, or waiting for a resource that is not available.

This could also be caused by a performance problem. Perhaps it is because your system is operating near the limits of its capacity.

Operating system performance problems, for both z/VSE and VM, are a specialized area requiring detailed analysis. Investigate these problems with the assistance of your z/VSE or VM systems programmer.

A performance problem may 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 manifest itself as a problem that occurs only when specific queues are accessed.

The following symptoms might indicate that WebSphere MQ is running slowly:

- Your system is slow to respond to WebSphere MQ commands.
- Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem may lie with WebSphere MQ for z/VSE itself. If you suspect this, you need to contact your IBM Support Center for assistance.

Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making WebSphere MQ calls are discussed in the following sections.

For more information about application design, see the *WebSphere MQ Application Programming Guide*.

Effect of message length

Although WebSphere MQ allows messages to hold up to 4 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, you should send only the essential data in a message; for example, in a request to debit a bank account, the only information that may need to be passed from the client to the server application is the account number and the amount of the debit.

Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message, correlation, and group identifiers (*MsgId*, *CorrelId*, and *GroupId*) in the message descriptor to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the *BufferLength* field set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the *MaxMsgLength* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be as high as 4 MB, the maximum allowed by WebSphere MQ for z/VSE.

Use of the MQPUT1 call

Use the MQPUT1 call only if you have a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

Incorrect output

The term "incorrect output" can be interpreted in many different ways. For the purpose of problem determination within this book, the meaning is explained in "Have you obtained incorrect output?" on page 204.

Two types of incorrect output are discussed in this section:

- Messages that do not appear when you are expecting them.
- Messages that contain the wrong information, or information that has been corrupted.

Additional problems that you might find if your application includes the use of distributed queues are also discussed.

Messages that do not appear on the queue

If messages do not appear 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, are the queue and maximum message length sufficiently large?
  - Is the queue enabled for putting?
  - Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Are you able to get any messages from the queue?
  - Do you need to take a syncpoint?

    If messages are being put or retrieved within syncpoint, 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. You should ensure that you are waiting long enough for a response.
  - Are you waiting for a specific message that is identified by a message, correlation, or group identifier (*MsgId*, *CorrelId*, or *GroupId*)?

    Check that you are waiting for a message with the correct *MsgId*, *CorrelId*, or *GroupId*. A successful MQGET call sets both these values to that of the message retrieved, so you may 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?
  - Has another application got exclusive access to the queue?

If you are unable to find anything wrong with the queue, and WebSphere MQ is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application get started?

  If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Did the application complete correctly?

  Look for evidence of an abnormal end on the system log and z/VSE console.
- 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 multi-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 this is the case, refer to "Messages that contain unexpected or corrupted information."

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 points:

- 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 may 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 corrupt to the other.
- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not really intended for an application servicing a different queue.

  If your application has used an alias queue, check that the alias points to the correct queue.
- Has the trigger information been specified correctly for this queue?

  Check that your application should have been started; or should a different application have been started?

If these checks do not enable you to solve the problem, you should 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, you should also consider the following points:

- Has 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 WebSphere MQ. Check that the connection between the two systems, and the channels between the two queue managers, are active.
- Is triggering set on in the sending system?
- Is the message you are waiting for a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

  This could mean that an application was unable to put the required message onto the queue. If this is 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 the *WebSphere MQ Application Programming Reference* manual 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 stops the distributed queuing component. See the *WebSphere MQ Intercommunication* book for more information about distributed queuing.

System log

WebSphere MQ uses the SYSTEM.LOG queue defined in the global system definition as its primary message log and additional informational messages are output to the z/VSE console. Typically, these detail starting, stopping, and initializing WebSphere MQ for z/VSE

If the SYSTEM.LOG queue is unavailable, the messages are directed to the CICS CSMT log. These messages should always be reviewed carefully for any error messages. The type of messages included in the SYSTEM.LOG queue can now be controlled by using the 'Log and Trace Settings'. Refer to “Queue Manager Log and Trace Settings” on page 88 for details.

You can view the contents of the system log using the Master Terminal transaction (MQMT) option 4 (Browse Queue Records).

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:33:55                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.LOG
    QSN Number : 00000001       LR-        0, LW-      249, DD-MQFLOG
                     Queue Data Record
Record Status : Written.        PUT date/time  : 20061120084057
Message Size  : 00000711        GET date/time  :

MQI000000I  PRG:MQPINIT1 TRN:MQSE TRM:A000 TSK:00043 11/20/2006 08:40:57
Queue manager started




EIBFN:   1A04       EIBRCODE:  000000000000 EXEC LINE: 000000
EIBRESP: 00000000   EIBRESP2:  00000000     EIBRSRCE:            ABCODE:


Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                                     PF11=MD    PF12=Explain
```

Figure 75. Browsing the system log

PF12 provides an explanation of messages written to the system log.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:38:44                  Resolution Information                     CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.LOG
    QSN Number : 00000001       LR-        0, LW-      249, DD-MQFLOG



Message Code 000000I Extended LOG  Description     Page      1
Queue manager started
Reason       :  The local queue manager has been started.
User Action  :  -- none --
System Action:  The queue manager is available for queuing
                services.




Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down                  PF11=MD    PF12=Queue
```

Figure 76. Browsing the system log - explain

Dead-letter queue

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by using the MQMT transaction. If the queue contains messages, you can use the browse facility to browse messages on the queue using the MQGET call.

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 may occur if you do not have a dead-letter queue on each queue manager you are using.

You can view the contents of the dead-letter queue using the Master Terminal transaction (MQMT) option 4 (Browse Queue Records).

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:44:01                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.DEAD.LETTER.QUEUE
    QSN Number : 00000001       LR-        0, LW-        2, DD-MQFERR
                      Queue Data Record
Record Status : Written.        PUT date/time  : 20061208125454
Message Size  : 00000372        GET date/time  :                       ASCII
Offset ....+....|....+....|....+....|....+....|....+....|....+....|....+....|
 00000 TEST MSG WITH MQMD VERSION 1 (with ASCII codepage)
 00070
 00140                                                          <END>









Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
               PF7=Up      PF8=Down    PF9=Hex       PF10=DLH   PF11=MD
```

Figure 77. Browsing the dead-letter queue

Note: The message size includes the 172 bytes for the dead-letter header (MQDLH) data structure. The "ASCII" above the scale line indicates that the message data has been converted to EBCDIC for display purposes. This is only done when the dead-letter message is ASCII and the message format is MQSTR.

PF10 displays the dead-letter header of the message.

```
12/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:50:11                  Browse Queue Records                       CIC1
MQWDISP                     SYSTEM IS ACTIVE                         A000

    Object Name: SYSTEM.DEAD.LETTER.QUEUE
    QSN Number : 00000001       LR-        0, LW-        2, DD-MQFERR
                      Dead-letter header
Reason  . . . . . . 2085 MQRC_UNKNOWN_OBJECT_NAME
Destination Q . . . BAD.QUEUE.FORCE.TO.DLQ
Destination QM. . . TS212.QM.PTHVSEA
Original Encoding .   546
Original CCSID. . .   850
Original Format . . MQSTR
Put appl type . . . WebSphere MQ for z/VSE
Put appl name . . . TSMQ300 MQ01
Put date. . . . . . 2006/12/08
Put time. . . . . . 12:54:54



Information displayed.
    5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Quit    PF4=Next      PF5=Prior
                                       PF9=Hex       PF10=Queue PF11=MD
```

Figure 78. Browsing the dead-letter queue - MQDLH

Using WebSphere MQ trace

WebSphere MQ for z/VSE relies on the CICS auxiliary trace for problem determination. To reduce overhead in a production environment, the trace points are not issued unless specified using the 'Log and Trace Settings' screen. Tracing should only be used when requested by IBM service personnel. Refer to “Queue Manager Log and Trace Settings” on page 88 for details.

Problem determination with clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are now additional reason codes for error conditions associated with 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 and receives the response MQRC_Q_MQR_NOT_AVAILABLE. An error message, written to the client log file, explains the cause of the error. Messages may also be logged at the server depending on the nature of the failure.

Terminating clients

Even though a client has terminated it is still possible for the process at the server to be holding its queues open. Normally, this will only be for a short time until the communications layer detects that the partner has gone.

Error messages with clients

When an error occurs with a client system, error messages are put into the error files associated with the server, if possible. If an error cannot be placed there, the client code attempts to place the error message in an error log in the root directory of the client machine.

OS/2 and UNIX systems clients

Error messages for OS/2 and UNIX systems clients are placed in the error logs on their respective WebSphere MQ server systems. Typically, these files appear in the QMGRS\@SYSTEM\ERRORS directory.

DOS and Windows clients

The location of the log file AMQERR01.LOG is set by the MQDATA environment variable. The default location, if not overridden by MQDATA, is:

```
C:\
```

Working in the DOS environment involves the environment variable MQDATA.

This is the default library used by the client code to store trace and error information; it also holds the directory name in which the qm.ini file is stored. (needed for NetBIOS setup). If not specified, it defaults to the C drive.

The names of the default files held in this library are:

AMQERR01.LOG
For error messages.

AMQERR01.FDC
For First Failure Data Capture messages.

Problems with SSL enabled channels

SSL enabled channels can fail for a number of reasons. Sometimes a channel will fail because it is meant to fail. That is, the remote system provided an X.509 certificate that did not match expected identifiers. Specifically, the partner certificate failed to match the SSL Peer Attributes parameter of the local channel.

In this case, a message is logged to the SYSTEM.LOG and the channel is terminated. This is the correct behavior in such a situation, as the channel should not continue when the remote system does not identify itself as expected.

SSL enabled channels can also fail due to problems with the WebSphere MQ, z/VSE, or the remote system environment. Specifically, an SSL enabled channel can fail due to:

- SSL availability.
- Cipher specification support.
- Client authentication failure.
- General channel failure.

SSL availability

It is essential for SSL enabled channels that the SSL feature is installed and available on both the local and remote systems participating in an SSL secured conversation.

WebSphere MQ for z/VSE initializes the SSL environment at system startup, if the an SSL key-ring sublibrary name is specified in the queue manager's

communication settings. At startup, WebSphere MQ will create an initialization instance for use by channels during normal system operation. The initialization instance is a data structure provided by the SSL service, and is associated with the SSL key-ring sublibrary.

The SSL initialization instance is anchored to an WebSphere MQ control block and is used each time an SSL enabled channel is activated. If the instance is invalid, all SSL enabled channels will fail during initialization, with the following error message:

```
006100E TCP/IP SSL INITIALIZATION FAILED
```

This particular error can also occur if the SSL key-ring member name does not identify valid private key and certificate files in the SSL key-ring sublibrary.

In addition, and in regard to the remote system, this error can occur if the remote system does not have the SSL feature installed and available, or the cipher specification required by the SSL client is not supported by the SSL server (that is, the receiver MCA).

If this message is encountered, the following should be checked:

- The SSL feature is installed and available.
- The SSL key-ring sublibrary name specified in the queue manager 'Communication Settings' identifies the correct sublibrary name.
- The SSL key-ring member name, also specified in the queue manager 'Communication Settings' identifies the member name of valid private key and certificate files.
- The remote system has SSL installed and available.
- The remote system supports the cipher specification identified by Sender channel.
- The remote system is configured to provide an X.509 certificate during SSL initial negotiation.

Following these checks, if the channel continues to fail, problem determination relating to channels in general should be pursued.

Cipher specification support

AN SSL enabled channel can fail if the SSL Cipher Specification identified by the Sender channel is not supported by the remote system, or if the SSL negotiations complete for a specification that is not identified by the Receiver channel.

Both of these cases produce the following error message:

```
006101E TCP/IP SSL CIPHER SPECIFICATION ERROR
```

For Sender channels, it is important that the cipher specification identified in the channel definition is supported by the remote system. The TCP/IP Administrator of the remote system should be able to provide a list of supported ciphers. For WebSphere MQ for z/VSE, the cipher is identified by a two-character code, (for example, 08, 09, 0A, 62). SSL for z/VSE documentation should be reviewed to determine what ciphers these codes identify and whether or not the remote system can support them.

For Receiver channels, it is essential that the cipher identified in the channel definition matches the cipher stipulated by the matching Sender channel definition. For example, if the Sender channel specifies a cipher code of '0A', then the Receiver channel must also specify '0A'. For Receiver channels, WebSphere MQ

completes the SSL negotiation and then checks that the agreed cipher matches the channel definition. If not, the channel is terminated.

Client authentication failure

The SSL client authentication parameter, specified on a channel definition, instructs WebSphere MQ to check that the remote partner exchanged an X.509 certificate during SSL negotiation. SSL negotiation occurs when the channel is initialized.

If a channel is configured to require a certificate from the remote system and one is not received, then a client authentication error will occur. Since the server, or WebSphere MQ receiver MCA, always provides a certificate during SSL negotiation, this error pertains to Sender channels only.

A client authentication failure produces the following error message:

```
006103E TCP/IP SSL CLIENT AUTHENTICATION ERROR
```

However, a client authentication error cannot occur with the current SSL for z/VSE service, because WebSphere MQ for z/VSE always requires a certificate from the client during SSL negotiation. If a certificate is not exchanged, WebSphere MQ will terminate the channel with an SSL initialization error, not a client authentication error.

General channel failure

It is possible for an SSL enabled channel to successfully establish an active SSL connection with a remote system and subsequently fail. In such a case, the channel has probably failed for any of the reasons applicable to channels in general.

When a TCP/IP connection between two systems (or between the queue manager and a remote client program) is established, WebSphere MQ will attempt to secure the connection using SSL services if the Sender (or client) channel is SSL enabled. This process may be successful, but subsequent channel negotiations over the secure connection fail. This is not an SSL channel failure.

If a channel fails due to reasons applicable to channels in general, problem determination relevant to general channel failure should be pursued.

Chapter 7. Message data conversion

Application data is converted within an application program when the MQGMO-CONVERT option is specified in the Options field of the MQGMO structure passed to an MQGET call, and all of the following are true:

- The code page or encoding fields set in the MQMD structure associated with the message on the queue differ from the code page or encoding fields set in the MQMD structure specified on the MQGET call.
- The format field in the MQMD structure associated with the message is not MQFMT-NONE.
- The buffer length specified on the MQGET call is not zero.
- The message data length is not zero.
- The queue manager supports conversion between the code page and encoding fields specified in the MQMD structures associated with the message and the MQGET call. LE/VSE is used for application data conversion and must have the appropriate code converters available. See “Using LE/VSE for conversion” on page 218 for more information.

The queue manager supports conversion of a number of data formats. If the format field of the MQMD structure associated with the message is one of the built-in formats, the queue manager can convert the message. If the format is not one of the built-in formats, you must write a data conversion exit program to convert the message.

When you move messages between systems, sometimes it is necessary to convert the application data into the character set and encoding required by the receiving system. Conversion can be done either from within application programs on the receiving system, or by the Message Channel Agents (MCAs) on the sending system. If data conversion is supported on the receiving system, we recommend that you use application programs to convert the application data, rather than depending on the data already being converted at the sending system.

If the sending MCA is to convert the data, the 'Convert Msgs' field must be set to Y on the definition of each sender channel for which conversion is required. If conversion fails on the sender channel, the message remains on the transmission queue and the channel is shut down, with a GET error shown on the system log.

The following built-in formats are converted by WebSphere MQ for z/VSE:

MQFMT_STRING
MQFMT_DEAD_LETTER_HEADER
MQFMT_TRIGGER
MQFMT_ADMIN
MQFMT_EVENT
MQFMT_PCF
MQFMT_EMBEDDED_PCF
MQFMT_REF_MSG_HEADER
MQFMT_IMS
MQFMT_IMS_VAR_STRING
MQFMT_COMMAND_1
MQFMT_COMMAND_2
MQFMT_SAP
MQFMT_RF_HEADER

MQFMT_RF_HEADER_2
MQFMT_CICS
MQFMT_DIST_HEADER

See the *WebSphere MQ Application Programming Reference* manual for more details regarding data conversion.

Data conversion exit programs

For application defined formats that do not conform to the built-in formats, conversion can be performed by an exit program.

The name of the exit program must be the same as your data format name. For example, if a message has the format (FMT_TEST), the exit program must be called FMT_TEST. WebSphere MQ checks that the format is not one of the built-in formats, then, if it is not, calls the exit program.

To build a user exit program for your own format:

1. Start with the supplied source skeleton DCHFMT4.
2. Follow the instructions in the prolog of DCHFMT4, ensuring that the correct macros are called to convert your structure.
3. Rename the program to your data format name.
4. CICS translate, compile, prelink and link-edit your program.
5. Place your exit program in a CICS application program library. Define the program to CICS in the usual way.

See the *WebSphere MQ Application Programming Guide* for more details regarding data conversion exit programs.

Using LE/VSE for conversion

For application message code page conversion, WebSphere MQ for z/VSE uses the Language Environment (LE) code set conversion facilities in a similar manner to the WebSphere MQ server. LE/VSE provides a number of supplied code converters, and facilities to build code converters that are not provided.

If you need code conversion for pages that are not provided by LE/VSE, you can edit the appropriate source code modules and build the converters. You also need to inform WebSphere MQ for z/VSE of the type number and the encoding of the user-defined code page.

Selecting 4 on the Configuration menu allows you to add user-defined code pages, and their type and encoding.

As well as the MQServer SBCS conversion, support is provided for:

- DBCS code pages.
- Mixed code pages.
- EUC code pages.
- ISO code page.
- Unicode (UCS-2 and UTF-8) code pages.

See “Code page conversion” on page 1011 for more details on LE/VSE code pages. The same LE code set conversion facilities are used for application data conversion.

We also strongly recommended that you read the section on code page conversion in the *C Run-time Programming Guide*.

Note: The following LE code pages have the equivalent numerics on WebSphere MQ for z/VSE.

ISO8859-1 - 819

IOS8859-7 - 813

ISO8859-9 - 920

IBM-eucJP - 33722

Building a conversion exit program

WebSphere MQ for z/VSE is shipped with a sample exit program (DCHFMT4). This sample provides conversion for standard data types recognized by supplied conversion macros. The following data structure encapsulates these supported data types:

```
struct testall
{
  MQBYTE  byte2[2];
  short   short2;
  MQLONG  long4;
  MQCHAR  char5[5];
  MQCHAR  charV;
};
```

This contains all the types of data (char, variable char, byte, long and short) that can be converted.

The following code in DCHFMT4 converts the structure shown earlier.

```
ConvertByte(2);     /* 3 byte binary data              */

AlignShort();
ConvertShort(1);    /* 2 byte integer                  */

AlignLong();
ConvertLong(1);     /* 4 byte integer                  */

ConvertChar(5);     /* 5 byte character data           */

ConvertVarChar();   /* Variable length character data  */
```

Note the use of AlignShort/Long before ConvertShort/Long. In z/VSE, there is no CRTMQCVX utility to create this code fragment for you.

These functions are defined using the following include files (see the DCHFMT4 source for a full listing):

```
#include <cmqc.h>            /* For MQI data types
#include <cmqxc.h>           /* For MQI exit-related definitions
#include <mqidatcv.h>        /* For sample macro definitions
```

The following linkage parameters are required to build a conversion exit program:

```
PHASE DCHFMT4,*
INCLUDE DCHFMT4
INCLUDE MQPDATCU
INCLUDE MQPDATCV
INCLUDE DFHELII
```

Because the exit program runs under CICS, a PPT entry is required in CICS for the program. For example:

```
DEFINE  PROGRAM(DCHFMT4) GROUP(MQM) LANGUAGE(C)
```

Note: The user exit program must be contained in the same CICS region as WebSphere MQ — it cannot be placed in a separate region for dynamic routing.

Chapter 8. Programmable system management

This chapter describes Instrumentation Events and WebSphere MQ PCFs (Programmable Command Formats) and their relationship to other parts of the WebSphere MQ products.

Instrumentation events

In WebSphere MQ, an instrumentation event is a logical combination of conditions 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.

WebSphere MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can, therefore, use these events to monitor the operation of queue managers.

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:

Gets the message from the queue.
Processes the message to extract the event data.

WebSphere MQ instrumentation events may be categorized as follows:

Queue manager events.
Channel events.
Performance events.
Command events.
Configuration events.

Queue manager events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached.

Channel events are reported by channels as a result of conditions detected during their operation. For example, when a channel instance is stopped.

Command events are notifications that an MQSC or PCF command has run successfully.

Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

SSL events are a type of channel event. 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.

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. Event queues are configurable using MQMT option 1.1, PF11. This activates the

following screen:

```
11/23/2010        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:24:13                   Global System Definition                 CIC1
MQWMSYS                          Event Settings                     A000

    Event queues
    Queue manager events : SYSTEM.ADMIN.QMGR.EVENT
    Channel events . . . : SYSTEM.ADMIN.CHANNEL.EVENT
    Performance events . : SYSTEM.ADMIN.PERFM.EVENT
    Command events . . . : SYSTEM.ADMIN.COMMAND.EVENT
    Configuration events : SYSTEM.ADMIN.CONFIG.EVENT

    Qmgr events          Channel events      Performance events
    Inhibit  . . . . : N  Started  . . . : Y  Queue depth  . . : N
    Local  . . . . . : N  Stopped  . . . : Y  Service interval : N
    Remote . . . . . : N  Conversion err : N
    Authority  . . : N  Auto-define. . : N
    Start/Stop . . : Y  SSL  . . . . . : N
    Command  . . . : Y
    Configuration  : N


Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 79. Global System Definition

The default names for the event queues are as follows:

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.COMMAND.EVENT	Command events[1]
SYSTEM.ADMIN.CONFIG.EVENT	Configuration events[1]
Note: 1. Event queue name is not configurable.	

By incorporating these events into your own system management application, you can monitor the activities across many queue managers, across many different nodes, for multiple WebSphere MQ applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support WebSphere MQ events).

Instrumentation events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator.

Review section “Features” on page 15 for prerequisites for this feature.

Queue manager events

These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

The event messages for queue manager events are put on the SYSTEM.ADMIN.QMGR.EVENT queue. The following queue manager event types are supported:

Inhibit
Local
Remote
Authority
Start/Stop

For each event type in this list, there is a queue manager attribute that enables or disables the event type. The conditions that give rise to the event (when enabled) include:

- 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.

  Note that a similar condition may occur during the internal operation of a queue manager, for example, when generating a report message. The reason code in an event message may match an MQI reason code, even though it is not associated with any application. Therefore you should 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 the processing of 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.

Inhibit events

Inhibit events indicate that an MQPUT or MQGET operation has been attempted against a queue, where the queue is inhibited for puts or gets respectively.

There are two types of Inhibit event:

Get Inhibited
Put Inhibited

Local events

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, when an application attempts to access an object that has not been defined.

There are three types of Local event:

Alias Base Queue Type Error.
Unknown Alias Base Queue.
Unknown Object Name.

Remote events

Remote events indicate that an application (or the queue manager) cannot access a (remote) queue on another queue manager. For example, when the transmission queue to be used is not correctly defined.

Remote events include the following:

Remote Queue Name Error.
Transmission Queue Type Error.
Transmission Queue Usage Error.
Unknown Default Transmission Queue.
Unknown Remote Queue Manager.
Unknown Transmission Queue.

Authority events

Authority events indicate that an authorization violation has been detected.

For example, an application attempts to open a queue for which it does not have the required authority, or a command is issued from a user ID that does not have the required authority.

There are four types of Authority event supported by WebSphere MQ: for VSE:

Not Authorized (type 1) - Connection failures.
Not Authorized (type 2) - Open failures.
Not Authorized (type 3) - Close failures.
Not Authorized (type 4) - Command failures.

Start and stop events

Start and stop events indicate that a queue manager has been started or has been requested to stop.

Start and Stop events include the following:

Queue Manager Active.
Queue Manager Not Active.

Channel events

These events are reported by channels as a result of conditions detected during their operation. For example, when a channel instance is stopped. Channel events are generated:

- By a command to stop a channel.
- When a channel instance starts or stops. Note that client connections do not cause Channel Started or Channel Stop events.
- When a channel receives a conversion error warning when getting a message.
- When a channel auto-definition event occurs. An auto-definition event occurs if the channel auto-definition feature is enabled and a connection request is received for a receiver or server connection channel that does not exist.
- When 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.

Channel event messages are put onto the SYSTEM.ADMIN.CHANNEL.EVENT queue, if it is available, and the queue manager is configured to generate channel events. Otherwise, they are ignored.

Channel events include the following:

Channel Started.
Channel Stopped.
Channel Conversion Error.
Channel auto-definition.
Channel SSL Error event.

Performance events

These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached.

Performance events are related to conditions that can affect the performance of applications that use a specified queue.

The event type is returned in the command identifier field in the message data.

Performance events are not generated for the event queues themselves. If a queue manager attempts to put a queue manager or a performance even message on an event queue and an error that would normally create an event is detected, another event is not created and no action is taken.

MQGET calls and MQPUT calls within a unit of work can cause performance related events to occur regardless of whether the unit of work is committed or backed out.

There are two types of performance event:

- Queue depth events, which include:
  Queue Depth High
  Queue Depth Low
  Queue Full
- Queue service interval events, which include:
  Queue Service Interval High
  Queue Service Interval OK

WebSphere MQ for z/VSE supports queue depth and service internal events for local queues, including transmissions queues.

As already stated, performance events are related to conditions that can affect the performance of applications that use a specified queue.

The scope of performance events is the queue, so that MQPUT calls and MQGET calls on one queue do not affect the generation of performance events on another queue.

Note: A message must be either put on, or removed from, a queue for any performance event to be generated.

The event data contains a reason code that identifies the cause of the event, a set of performance event statistics, and other data.

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. Also, the event data contains statistics related to the event.

You can use these statistics to analyze the behavior of a specified queue. The following table summarizes the event statistics. All the statistics refer to what has happened since the last time the statistics were reset.

Parameter	Description
TimeSinceReset	The elapsed time since last reset.
HighQDepth	Maximum queue depth since last reset.
MsgEnqCount	Messages put since last reset.
MsgDeqCount	Messages got since last reset.

Performance event statistics are reset when any of the following occur:
A performance event occurs.
A queue manager stops and restarts.

Queue depth events

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, it can be a 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 may 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.

Clearly, having advanced warning that problems may be on their way makes it easier to take preventive action. For this purpose, queue depth event are provided.

Queue depth events are related to the queue depth, that is, the number of messages on the queue. The types of queue depth events are:

- Queue Depth High events, which indicate that the queue depth has increased to a predefined threshold called the Queue Depth High limit.
- Queue Depth Low events, which indicate that the queue depth has decrease to a predefined threshold called the Queue Depth Low limit.
- Queue Full events, which 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 should take some preventive action. If this action is successful and the queue depth drop to a “safe” level, the queue manager can be configured to generate a Queue Depth Low event indicating an 'all clear' state.

Queue service interval events

Queue service interval events indicate whether a queue was “serviced” within a user-defined time interval called the service interval. Depending on the circumstances at your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

There are two types of queue service interval event:

- A Queue Service Interval OK event, which indicates that, following an MQPUT call or an MQGET call that leaves a non-empty queue, an MQGET call was performed within a user-defined time period, known as the service interval.
- A Queue Service Interval High event, which indicates that, following an MQGET or MQPUT call that leaves a non-empty queue, an MQPUT or an MQGET call was not performed within the user-defined service interval.

To enable both Queue Service Interval OK and Queue Service Interval High events you need to 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.

These events are mutually exclusive, which means that if one is enabled the other is disabled. However, both events can be simultaneously disabled.

In terms of queue service interval events, these are the possible outcomes:

- If the elapsed time between the put and get is less than or equal to the service interval:

  A Queue Service Interval OK event is generated, 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, if queue service interval events are enabled

Service interval 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 one or other of the queue service interval events is enabled.

The service timer measures the elapsed time between an MQPUT call to an empty queue or an MQGET call and the next put or get, provided the queue depth is nonzero between these two operations.

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.

The service timer is always reset after an MQGET call. It is also reset by an MQPUT call to an empty queue. However, it is not necessarily reset on a queue service interval event.

Following an MQGET call 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 the operation is an MQGET call 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.

Command events

Command events are notifications that an MQSC or PCF command has run successfully.

The event data contains this 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 using the administrator panels.

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 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. If using the administrator panel, the equivalent MQSC command is generated.

Every command event message that is generated is placed on the command event queue, SYSTEM.ADMIN.COMMAND.EVENT.

Command event generation

A command event message is generated in these situations:

- When the CMDEV queue manager attribute is specified as ENABLED and an MQSC or PCF command (or administrator panel) 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 these 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 these 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.
- When the queue manager is not active, no command event messages are generated since the queue manager is required to be active to put event messages to the event queues.

Command event usage

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 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.

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 three types of configuration events:
- Create object events.
- Change object events.
- Delete object events.

The event data contains this 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 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 the change.

Every configuration event message that is generated is placed on the queue, SYSTEM.ADMIN.CONFIG.EVENT.

The WebSphere MQ for z/VSE administrator panels can update configuration of objects without the queue manager being active. When CONFIGEV=Y, the user is not allowed to do any updates. An error message is displayed.

Configuration event generation

A configuration event message is put to the configuration event queue when the CONFIGEV queue manager attribute is ENABLED and any of these are true:

- Any of these commands, or their PCF equivalent, are issued:
  DELETE CHANNEL
  DELETE NAMELIST
  DELETE QMODEL/QALIAS/QREMOTE
- Any of these commands, or their PCF equivalent, are issued even if there is no change to the object:
  DEFINE/ALTER CHANNEL
  DEFINE/ALTER NAMELIST
  DEFINE/ALTER QMODEL/QALIAS/QREMOTE
  ALTER QMGR, unless the CONFIGEV attribute is DISABLED and is not changed to ENABLED
- Any of these 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 these 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.
- For the configuration event queue SYSTEM.ADMIN.CONFIG.EVENT.

Configuration event usage

Use configuration events to obtain information about your system, and to understand the factors that can affect your use of configuration events.

You can use configuration events 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.
- 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 share the same correlation identifier in their message descriptor.

For an MQSET call, or for any of these 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.

Enabling and disabling events

All instrumentation events must be enabled before they can be generated. You can enable and disable events by specifying the appropriate values for queue manager or queue attributes (or both) depending on the type of event. You do this using:
- PCF commands.
- MQSC (WebSphere MQ) commands.
- MQMT Master Terminal transactions.

Enabling queue manager events

Queue manager events can be enabled or disabled by changing queue manage attributes. This is possible using the master terminal transaction, MQMT option 1.1, PF11. This option displays the queue manager's event settings, where each event type can be enabled or disabled.

Alternatively, the queue manager attributes can be set using PCF (see “Programmable command formats” on page 235) or WebSphere MQ commands (see Chapter 9, “WebSphere MQ commands,” on page 507).

Enabling channel events

Channel events are enabled via the queue manager event settings,

however, channel events can be suppressed by not defining the channel events queue, or by making it put-inhibited. Note that this could cause a queue to fill up if remote event queues point to a put-inhibited channel events queue.

If a queue manager does not have a SYSTEM.ADMIN.CHANNEL.EVENT queue, or if this queue is put inhibited, all channel event messages are discarded unless they are being put by an MCA across a link to a remote queue. In this case they are put on the dead-letter queue.

Enabling performance events

Performance events as a whole must be enabled on the queue manager, otherwise no performance events can occur. You can then enable the specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event.

Queue specific event settings are available via the local queue extended definition, available via MQMT option 1.2.

```
2011/10/10        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
08:21:54               Queue Extended Definition                    CIC1
MQWMQUE                                                             A001

Object Name: MY.XMITQ

General               Maximums                   Events
Type   . . : Local    Max. Q depth . : 00010000  Service int. event: N
File name  : MQFI001  Max. msg length: 00040000  Service interval  : 00000000
Usage  . . : T        Max. Q users . : 00000100  Max. depth event  : N
Shareable  : Y        Max. gbl locks : 00000100  High depth event  : N
Dist.Lists : N        Max. lcl locks : 00000100  High depth limit  : 000
PropCtl. . : C                                   Low depth event . : N
Triggering                                       Low depth limit . : 000
Enabled  . : Y        Transaction id.:
Type . . . : E        Program id . . : MQPSEND
Max. starts: 0001     Terminal id  . :
Restart  . : N        Channel name . : MY.SDR
User data  :
           :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
            PF9=List  PF10=Queue
```

Figure 80. Extended definition

Enabling queue depth events

By default, all queue depth events are disabled. To configure a queue for any of the queue depth events you must:

Enable performance events on the queue manager.
Enable the event on the required queue.
Set the limits, if required, to the appropriate levels, expressed as a percentage of the maximum queue depth.

Enabling queue service interval events

To configure a queue for queue service interval events you must:

Enable performance events on the queue manager, for example, using the queue manager attribute PerformanceEvent (PERFMEV in MQSC).
Set the control attribute, QServiceIntervalEvent, for a Queue Service Interval High or OK event on the queue, as required (QSVCIEV in MQSC).

Specify the service interval time by setting the QServiceInterval attribute for the queue to the appropriate length of time (QSVCINT in MQSC).

For 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)
```

Note: When enabled, a queue service interval event can only be generated on an MQPUT call or an MQGET call. The event is not generated when the elapsed time becomes equal to the service interval time.

Automatic enabling of queue service interval events

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.

Similarly, when an OK event is generated on a queue, the queue manager automatically disables OK events and enables high events for that queue.

Event queues

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 format of messages required for transmission queues.

Note: Attributes related to events for queues can be modified using the MQSET MQI call, commands, or the master terminal transactions.

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 will be lost. The event messages are not, for example, saved on the dead-letter (undelivered-message) queue.

However, the event queue may be defined as a remote queue. Then, if there is a problem on the remote system putting messages to the resolved queue the event message will appear on the remote system's dead-letter queue.

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 happens whether the event message is put on the performance event queue or not.

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 may require that an operator be informed, other events may start off an application that performs some administration tasks automatically.

Format of event messages

Event messages contain information about the event and its origin. Typically, these messages are processed by a system management application program tailored to meet the requirements of the enterprise at which it runs. As with all WebSphere MQ messages, an event message has two parts:

- A message descriptor.
- The message data.

The message descriptor is based on the MQMD structure, which is defined in the WebSphere MQ Application Programming Reference manual. The message data is also made up of an event header and the event data. The event header contains the reason code that identifies the event type. The putting of the event message and any subsequent actions arising do not affect the reason code returned by the MQI call that caused the event. The event data provides further information about the event.

Message descriptor (MQMD) in event messages

The format of the message descriptor is defined by the WebSphere MQ MQMD dat structure, which is found in all WebSphere MQ messages and is described in the WebSphere MQ Application Programming Reference manual. The message descriptor contains information that can be used by a user-written system monitoring application. For example:

- The message type.
- The format type.
- The date and time that the message was put on the event queue.

In particular, the information in the descriptor informs a system management application that the message type is MQMT_DATAGRAM, and the message format is MQFMT_EVENT.

In an event message, many of these fields contain fixed data, which is 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 event message was put on the event queue.

Message data in event messages

The event message data is based on the programmable command format (PCF) which is used in PCF command inquiries and responses. The event message consists of two parts: the event header and the event data.

Event header

The MQCFH structure is the header structure used for event messages and for PCF messages. When the structure is used for event messages, the message descriptor Format field is MQFMT_EVENT. The data type of the

following parameters (MQLONG) is described in the WebSphere MQ Application Programming Reference manual.

Event data
The event message data contains information specific to the event that was generated. This data includes the name of the queue manager and, where appropriate, the name of the queue.

The data structures returned depend on which particular event was generated. In addition, for some events, certain parameters of the structures are optional, and are returned only if they contain information that is relevant to the circumstances giving rise to the event. The values in the data structures depend on the circumstances that caused the event to be generated.

Note: The PCF structures in the message data are not returned in a defined order. They must be identified from the parameter identifiers shown in the description.

Event messages

WebSphere MQ for z/VSE supports the following event messages:
- Alias Base Queue Type Error
- Channel auto-define
- Channel Conversion Error
- Channel SSL error
- Channel Started
- Channel Stopped
- Channel Stopped By User
- Command
- Configuration
- Get Inhibited
- Not Authorized (type 1)
- Not Authorized (type 2)
- Not Authorized (type 3)
- Not Authorized (type 4)
- Put Inhibited
- Queue Depth High
- Queue Depth Low
- Queue Full
- Queue Manager Active
- Queue Manager Not Active
- Queue Service Interval High
- Queue Service Interval OK
- Remote Queue Name Error
- Transmission Queue Type Error
- Transmission Queue Usage Error
- Unknown Alias Base Queue
- Unknown Object Name
- Unknown Remote Queue Manager
- Unknown Transmission Queue

For more information about events, see the chapter "Event monitoring" in *WebSphere MQ Monitoring WebSphere MQ*.

Programmable command formats

This section describes Programmable Command Format (PCF) messages in terms of their use, prerequisites, definitions, and binary structure.

Introduction to Programmable Command Formats (PCFs)

The problem PCF commands solve

The administration of distributed networks can become very complex. The problems of administration will 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 PCFs are

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: queue managers, process definitions, queues, and channels. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local or remote, via 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).

Preparing WebSphere MQ for PCF

WebSphere MQ for z/VSE requires several resources to be active and available before it can support PCFs. Specifically, WebSphere MQ requires:

- System command queue is defined and available.
- PCF command server is active in CICS.

System command queue

The system command queue (usually SYSTEM.ADMIN.COMMAND.QUEUE) is specified in the queue manager's global system definition, as a communication parameter.

The global system definition can be accessed using the MQMT transaction, option 1.1 (the system command queue is one of the queue manager's "Communication Settings" and is accessible using PF9):

```
2011/10/31        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
21:29:46                   Global System Definition                  CIC1
MQWMSYS                     Communications Settings                  A000

    TCP/IP settings                          Batch Interface settings
    Licensed clients . . : 00000             Batch Int. identifier: MQBSRV39
    Adopt MCA  . . . . . : N                 Batch Int. auto-start: Y
    Adopt MCA Check  . . : N
                                             Channel Auto-Definition
                                             Auto-definition  . . : N
    SSL parameters                           Auto-definition exit :
    Key-ring sublibrary  :
    Key-ring member  . . :
    SSL reset count  . . :

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : Y

Requested record displayed.
PF2=Return PF3=Quit PF4/Enter=Read PF6=Update PF10=Listeners PF11=Services
```

Figure 81. PCF parameters

PCF messages, to be processed by the z/VSE queue manager, should be placed on the queue identified by the system command queue parameter (the system reply queue parameter is used by the MQSC batch utility, and is not relevant in the current context).

PCF messages placed on the system command queue are processed by the PCF command server, transaction MQCS.

PCF command server

The PCF command server must be active in CICS before PCF messages can be processed by WebSphere MQ for z/VSE. The PCF command server (transaction MQCS) can be activated in two ways:

- Manually in native CICS.
- Automatically by the queue manager.

Starting the command server manually: The PCF command server can be manually started in native CICS by running the MQCS transaction. If successful, starting the command server this way will display the following message:

```
PCF Command Server started.
```

The following message will also be written to the WebSphere MQ system log (assuming the queue manager is configured to log informational messages, see "Queue Manager Log and Trace Settings" on page 88.

```
MQI007000I PCF command server started
```

If the command server fails to start, review the system log for associated error messages.

Stopping the command server manually: The PCF command server can be manually stopped in native CICS by running the MQCS transaction with the terminate ("X") parameter. For example:

```
MQCS X
```

The terminate parameter places a special stop request message on the system command queue. The command server, when it retrieves the stop request from the command queue, finishes processing and terminates. If successful, running MQCS with the terminate parameter will display the following message:

```
PCF Command Server termination requested.
```

If the stop request is successful, the command server stops and the following message is written to the system log:

```
MQI007017I PCF command server stopped
```

If the stop request fails, or the command server fails to stop, review the system log for associated error messages.

Starting the command server automatically: The PCF command server can be started automatically in CICS by configuring the queue manager to start the server when the queue manager is initialized. This is achieved by setting the command server auto-start option on.

The command server auto-start parameter is configurable as part of the queue manager's global system definition (MQMT option 1.1) and the communication settings (PF9). See Figure 81 on page 236.

The command server auto-start parameter can be set to (Y)es or (N)o. To activate the command server automatically when the queue manager is initialized, set the auto-start parameter to (Y)es.

The command server runs as transaction MQCS. The auto-activation of the command server can be verified by inquiring on the active tasks in CICS. If the command server is active, the MQCS transaction will be listed as an active task.

If the command server fails to start automatically, check the system log for associated error messages.

Stopping the command server automatically: The command server can be stopped automatically by stopping the queue manager (by running the MQST transaction).

The command server can be stopped automatically this way regardless of how the server was started. For example, if the command server was started manually in native CICS, it can still be stopped automatically by stopping the queue manager.

Using PCFs

This section describes how to use the PCFs in a systems management application program for WebSphere MQ remote administration. The topic includes:

- PCF command messages.

- Responses.
- Authority checking for PCF commands.

PCF command messages

Each command and its parameters are sent as a separate command message containing a PCF header followed by a number of parameter structures (see "MQCFH - PCF header" on page 486). 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.

The queue to which the PCF commands are sent is often called the SYSTEM.ADMIN.COMMAND.QUEUE. The command server servicing this queue sends the replies to the queue defined by the ReplyToQ and ReplyToQMgr fields in the message descriptor of the command message.

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

PCF commands should be placed on the system command queue (specified in the queue manager's global system definition). PCF response messages are place on the ReplyToQ specified in the message descriptor of the originating PCF command.

Message descriptor for a PCF command: The WebSphere MQ message descriptor is fully documented in the WebSphere MQ Application Programming Reference manual.

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

Report
Any valid value, as required.

MsgType
This 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 AS/400, OS/2, Windows NT, or UNIX systems, set this field to the encoding used for the message data; conversion will be performed if necessary.

CodedCharSetId
If you are sending to AS/400, OS/2, Windows NT, or UNIX systems, set this field to the coded character-set identifier used for the message data; conversion will be performed if necessary.

Format
Set to MQFMT_ADMIN.

Priority
Any valid value, as required.

Persistence
Any valid value, as required.

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

CorrelId
The sending application may 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 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 these 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: WebSphere MQ for z/VSE does not support user-defined PCF formats. Unlike some MQ platforms that support PCF messages with an MQMD format of MQFMT_PCF, the WebSphere MQ for z/VSE command server will reject any PCF message that does not have an MQMD Format of MQFMT_ADMIN.

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" on page 486 for details). The message identifier and correlation identifier are set according to the report options of the request.

If a single command specifies a generic object name, a separate response is returned in its own message for each matching object. For the purpose of 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.

Responses

There are three types of response, described below:

- OK response
- Error response
- Data response

OK response: This 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 may be followed by one or more warning parameter structures appropriate to this reason code.

In either case, for an inquire command further parameter structures may follow.

Error response: If the command has an error, one or more error response messages are sent (more than one may be sent even for a command which would normally only have 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 which 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:

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 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 occurs if a list structure (MQCFIL) sent with the command contains an invalid or duplicate selector. 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.

MQIA_CODED_CHAR_SET_ID
This 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 consists of an OK response (as described above) to an inquire command. The OK response is followed by additional structures containing the requested data.

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

For specific information about data response messages, refer to "Data responses to commands" on page 381.

Message descriptor for a response: A response message (obtained using the Get-message option MQGMO_CONVERT) has the following fields in the message descriptor, defined by the putter of the message. The actual values in the fields are generated by the queue manager:

MsgType
This is MQMT_REPLY.

MsgId This is generated by the queue manager.

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

Format
This 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.

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. 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 this is to have a matching user ID on both the local and remote systems.

Authority checking is implemented differently on each platform. For more information about PCF command, and command resource authority checking, refer to Chapter 12, "Security," on page 651.

Error codes applicable to all commands

In addition to those listed under each command format, any command may return the following in the response format header (descriptions of the MQRC_* error codes are given in the WebSphere MQ Application Programming Reference manual):

Reason (MQLONG)

The value can be:

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_NONE
(0, X'000') No reason to report.

MQRCCF_COMMAND_FAILED
Command failed.

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

MQRCCF_CFH_CONTROL_ERROR
Control option not valid.

MQRCCF_CFH_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
Message sequence number not valid.

MQRCCF_CFH_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFH_TYPE_ERROR
Type not valid.

MQRCCF_CFH_VERSION_ERROR
Structure version number is not valid.

MQRCCF_ENCODING_ERROR
Encoding error.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

MQRCCF_MSG_TRUNCATED
Message truncated.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

Definitions of the PCFs

This section contains reference material for the PCFs of commands and responses sent between an WebSphere MQ systems management application program and an WebSphere MQ queue manager.

WebSphere MQ for z/VSE supports these PCF commands:

- Change Channel
- Change Channel Listener
- Change Namelist
- Change Queue
- Change Queue Manager
- Change Service
- Copy Channel
- Copy Channel Listener
- Copy Namelist
- Copy Queue
- Copy Service
- Create Channel
- Create Channel Listener
- Create Namelist
- Create Queue
- Create Service
- Delete Channel
- Delete Channel Listener
- Delete Namelist
- Delete Queue
- Delete Service
- Escape
- Inquire Channel
- Inquire Channel Authentication Records
- Inquire Channel Names
- Inquire Channel Status
- Inquire Connection
- Inquire Channel Listener
- Inquire Channel Listener Status
- Inquire Namelist
- Inquire Namelist Names
- Inquire Queue
- Inquire Queue Manager
- Inquire Queue Names
- Inquire Queue Status
- Inquire Service
- Inquire Service Status
- Ping Queue Manager
- Reset Channel
- Set Channel Authentication Record
- Start Channel
- Start Service
- Start Channel Listener
- Start Channel Listener
- Stop Channel
- Stop Connection
- Stop Channel Listener
- Stop Service
- Change Subscription
- Copy Subscription
- Create Subscription
- Delete Subscription
- Inquire Subscription
- Inquire Subscription Status
- Change Topic
- Copy Topic

- Create Topic
- Clear Topic String
- Delete Topic
- Inquire Topic
- Inquire Topic Names
- Inquire Topic Status

Programmable constants and data structures for these commands and their parameters are available in copybooks and header files provided with WebSphere MQ for z/VSE as follows:

COBOL:

- CMQV.C
- CMQXV.C
- CMQCFBSL.C
- CMQCFBSV.C
- CMQCFV.C
- CMQCFHL.C
- CMQCFHV.C
- CMQCFILL.C
- CMQCFILV.C
- CMQCFINL.C
- CMQCFINV.C
- CMQCFSLL.C
- CMQCFSLV.C
- CMQCFSTL.C
- CMQCFSTV.C
- CMQCFXLL.C
- CMQCFXLV.C

C:

- CMQC.H
- CMQXC.H
- CMQCFC.H

PL/I:

- CMQP.P
- CMQXP.P
- CMQCFP.P

Note that the command descriptions in the following sections pertain to WebSphere MQ for z/VSE. When considering PCF messages intended for other platforms, you should also refer to WebSphere MQ Programmable System Management (SC33-1482-08).

Inquire commands can now specify an optional integer filter structure (MQCFIF) or string filter structure (MQCFSF). This allows you to filter the information displayed by one (and only one) of the attributes of objects.

Change Channel

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition.

This PCF is supported on all platforms.

For any optional parameters that are omitted, the value does not change.

Required parameters:

ChannelName, ChannelType

Optional parameters:

BatchInterval, BatchSize, ChannelDesc, ChannelMonitoring, ChannelStatistics, ConnectionName, DataConversion, DiscInterval, DiscRetryCount, LongRetryCount, LongRetryInterval, MaxMsgLength, MaxWait, MsgExit, MsgUserData, PortNumber, PropertyControl, ReceiveExit, ReceiveUserData, SecurityExit, SecurityUserData, SendExit, SendUserData, SeqNumberWrap, ShortRetryCount, ShortRetryInterval, SSLCipherSpec, SSLClientAuth, SSLPeerName, TpName, TransportType, XmitQName, XmitSize

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of channel being changed. The value may be:

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_SVRCONN
Client.

Optional parameters

BatchInterval (MQCFIN)

Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

This is the approximate time in milliseconds that a channel will keep a batch open, if fewer than BatchSize messages have been transmitted in the current batch.

If BatchInterval is greater than zero, the batch is terminated by whatever occurs first:

- BatchSize messages have been sent.
- BatchInterval milliseconds have elapsed since the start of the batch.

If BatchInterval is zero, the batch is terminated by whatever first:

- BatchSize messages have been sent.
- the transmission queue becomes empty.

BatchInterval must be in the range zero through 999 999 999.

This parameter applies only to channels with a ChannelType of: MQCHT_SENDER, or MQCHT_SERVER.

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that should be sent down a channel before a checkpoint is taken.

The batch size which is actually used is the lowest of:

- The BatchSize of the sending channel.
- The BatchSize of the receiving channel.

Specify a value in the range 1 through 999 999.

This parameter is not valid for channels with a ChannelType of MQCHT_SVRCONN.

ChannelStatistics (MQCFIN)
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL)

Specifies whether statistics data is to be collected and, if so, the rate at which the data is collected.

The value can be:
MQMON_OFF
Statistics data collection is turned off for this channel.
MQMON_Q_MGR
The value of the queue manager's ChannelStatistics parameter is inherited by the channel.
MQMON_LOW
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.
MQMON_MEDIUM
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.
MQMON_HIGH
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected.

The value can be:
MQMON_OFF
Online monitoring data collection is turned off for this channel.
MQMON_Q_MGR
The value of the queue manager's ChannelMonitoring parameter is inherited by the channel. This is the default value.
MQMON_LOW
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel. e of

MQMON_HIGH
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set, identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Specify the name of the machine as required for the stated TransportType:

- For MQXPT_LU62, specify the fully-qualified name of the partner LU.
- For MQXPT_TCP specify either the host name or the network address of the remote machine.

This parameter is valid only for ChannelType values of MQCHT_SENDER, MQCHT_SERVER and MQCHT_REQUESTER.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for ChannelType values of MQCHT_SENDER and MQCHT_SERVER.

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

Specify a value in the range 0 through 999 999.

This parameter is valid only for ChannelType values of MQCHT_SENDER and MQCHT_SERVER.

DiscRetryCount (MQCFIN)
Disconnection retry count (parameter identifier: MQIACH_DISC_RETRY).

Specifies the maximum number of retries that the channel waits for messages to be put on a transmission queue before terminating the channel. The retries occur at a frequency specified by the DiscInterval parameter.

Specify a value in the range 0 through 99 999 999.

This parameter is valid only for ChannelType values of MQCHT_SENDER and MQCHT_SERVER.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by ShortRetryCount has been exhausted, this specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by LongRetryInterval.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must subsequently be restarted with a command (it is not started automatically by the queue manager).

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for ChannelType values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the queue manager. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by ShortRetryCount has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for ChannelType values of MQCHT_SENDER or MQCHT_SERVER.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This is compared with the value for the remote channel and the actual maximum is the lowest of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment. For WebSphere MQ for z/VSEthe upper limit is 4 MB.

MaxWait (MQCFIN)
Maximum TCP/IP wait (parameter identifier: MQIA_RECEIVE_TIMEOUT).

Specifies the maximum number of seconds to wait for a remote response on an active TCP/IP channel (sender or receiver).

Specify a value in the range 0 through 999999. The value zero means the channel waits indefinitely for data.

MsgExit (MQCFSL)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is given the entire application message and message descriptor for modification.

For channels with a channel type (ChannelType) of MQCHT_SVRCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

The format of the string is the same as for SecurityExit.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

WebSphere MQ for v/VSE supports a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/VSE, you can specify the names of up to 8 exit programs.

MsgUserData (MQCFSL)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

For channels with a channel type (ChannelType) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

WebSphere MQ for z/VSE supports a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the MsgExit list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding
- An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/VSE, you can specify up to 8 strings.

PortNumber (MQCFIN)
TCP/IP port number (parameter identifier: MQIACH_PORT_NUMBER).

For TCP/IP channels only, specifies the port number (for example, 1414) corresponding to an MQ Listener process running on the host specified by the ConnectionName parameter.

This parameter is valid only for ChannelType values of MQCHT_SENDER, MQCHT_SERVER and MQCHT_REQUESTER.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies what happens to properties of messages when the message is about to be sent to a queue manager that does not understand the concept of a property descriptor. This parameter is applicable to Sender and Server channels. The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

ReceiveExit (MQCFSL)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for SecurityExit.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

WebSphere MQ for z/VSE supports a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- An individual string must not exceed MQ_EXIT_NAME_LENGTH.

- On z/VSE, you can specify the names of up to 8 exit programs.

ReceiveUserData (MQCFSL)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

WebSphere MQ for z/VSE supports a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the ReceiveExit list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/VSE, you can specify up to 8 strings.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a non-blank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.
- Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow. Any security message flows received from the remote processor on the remote machine are passed to the exit.

The format of the string is a 1-8 character CICS program name. The exit name must identify a program defined to the CICS region.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

SecurityUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit (MQCFSL)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for SecurityExit.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

WebSphere MQ for z/VSE supports a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure. v You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/VSE you can specify the names of up to 8 exit programs.

SendUserData (MQCFSL)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

WebSphere MQ for z/VSE supports a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the SendExit list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/VSE, you can specify up to 8 strings.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 through 999 999 999.

This parameter is not valid for channels with a ChannelType of MQCHT_SVRCONN.

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by ShortRetryInterval before the (normally longer) LongRetryCount and LongRetryInterval are used.

Specify a value in the range 0 through 999 999 999.

This parameter applies only to channels with a ChannelType of: MQCHT_SENDER, or MQCHT_SERVER.

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the queue manager. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for ChannelType values of MQCHT_SENDER or MQCHT_SERVER.

SSLCipherSpec (MQCFST)
SSL cipher specification (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

Specifies the SSL cipher specification to use when establishing an SSL enabled channel. Unlike other MQ platforms that expect a 32-character specification name, WebSphere MQ for z/VSEexpects a 2-character specification code.

For a list of valid cipher specification codes, refer to SSL product documentation.

The maximum length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

SSLClientAuth (MQCFIN)
SSL client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

Specifies whether or not a PKI certificate is required when establishing an SSL enabled channel.

The value can be:

MQSCA_REQUIRED
PKI certificate is required.

MQSCA_OPTIONAL
PKI certificate is optional.

SSLPeerName (MQCFST)
SSL peer name (parameter identifier: MQCACH_SSL_PEER_NAME)

Identifies certain characteristics that are expected to match the contents of a PKI certificate received when establishing an SSL enabled channel.

The maximum length of the string is MQ_DISTINGUISHED_NAME_LENGTH.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

This is the LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH.

This parameter is valid only for channels with a TransportType of MQXPT_LU62. It is not valid for receiver channels.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP/IP.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if ChannelType is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

XmitSize (MQCFIN)
Maximum transmission size (parameter identifier: MQIACH_MAX_XMIT_SIZE).

Specifies the maximum number of bytes that may be sent in a single protocol transmission.

For LU6.2 channels, specify a value between 476 and 32000.

For TCP/IP channels, specify a value between 476 and 65535.

Note that the optional parameters described in this section are also applicable to the following PCF commands:

- "Copy Channel" on page 285
- "Create Channel" on page 292

Error codes

In addition to the values for any command shown in "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry value is not valid.

MQRCCF_LONG_TIMER_ERROR
Long retry interval is not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry value is not valid.

MQRCCF_SHORT_TIMER_ERROR
Short retry interval is not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change Channel Listener

The Change Channel Listener (MQCMD_CHANGE_LISTENER) command changes the specified attributes of an existing WebSphere MQ listener definition.

For any optional parameters that are omitted, the value does not change.

Required parameters:
ListenerName, TransportType

Optional parameters:
Backlog, IPAddress, ListenerDesc, Port, StartMode

Required parameters

For details of the required parameters for Change Channel Listener, see "Create Channel Listener" on page 295.

Optional parameters

For details of the optional parameters for Change Channel Listener, see "Create Channel Listener" on page 295.

Change Namelist

The Change Namelist (MQCMD_CHANGE_NAMELIST) command changes the specified attributes of an existing WebSphere MQ namelist.

For any optional parameters that are omitted, the value does not change.

Required parameters:
NamelistName

Optional parameters:
NamelistDesc, Names

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME). The name of the namelist to be changed.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This is a plain-text comment that provides descriptive information about the namelist definition. It should contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGT.

Names (MQCFSL)
The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the Count field in the MQCFSL structure. The length of each name is given by the StringLength field in the that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Change Queue

The Change Queue (MQCMD_CHANGE_Q) command changes the specified attributes of an existing WebSphere MQ queue.

This PCF is supported on all platforms.

For any optional parameters that are omitted, the value does not change.

Required parameters:

QName, QType

Optional parameters:

BaseQName, CICSFileName, DefinitionType, InhibitGet, InhibitPut, MaxGlobalLocks, MaxLocalLocks, MaxMsgLength, MaxQDepth, MaxQTriggers, MaxQUsers, PropertyControl, QDepthHighEvent, QDepthHighLimit, QDepthLowEvent, QDepthLowLimit, QDepthMaxEvent, QDesc, QServiceInterval, QServiceIntervalEvent, QueueAccounting, QueueMonitoring, QueueStatistics, RemoteQMgrName, RemoteQName, Reorganization, ReorgStartTime, ReorgInterval, ReorgCatalog, Shareability, TriggerChannelName, TriggerControl, TriggerData, TriggerProgramName, TriggerRestart, TriggerTerminalId, TriggerTransactionId, TriggerType, Usage, XmitQName

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be changed. The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being changed.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Optional parameters

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a local or remote queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CICSFileName (MQCFST)
CSD file name for queue messages (parameter identifier: MQCA_CICS_FILE_NAME).

This is the name of a VSAM file defined to CICS that is to be associated with a queue definition for storing queue messages.

The maximum length of the string is MQ_CICS_FILE_NAME_LENGTH.

Note: CICSFileName must be specified with the Create Queue command, but it cannot be specified with the Change Queue command, that is, you cannot change the VSAM file that hosts a queue after it has been defined. To change the VSAM file, the queue must be deleted and redefined. The CICSFileName parameter is optional for the Copy Queue command.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

This attribute is valid for model queues only. For all other queue types the default is MQQDT_PREDEFINED.

For a model queue, The value can be:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

MaxGlobalLocks (MQCFIN)
Buffer size for queue manager to manage concurrent queue access (parameter identifier: MQIA_MAX_GLOBAL_LOCKS).

The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls, for each queue in the system, for recovery.

A value of 500 is normally sufficient.

The maximum value is 1000.

MaxLocalLocks (MQCFIN)
Buffer size for applications to manage concurrent queue access (parameter identifier: MQIA_MAX_LOCAL_LOCKS).

The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls for each queue and task for recovery.

A value of 500 is normally sufficient.

The maximum value is 1000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length for messages on the queue. Because applications may use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue, the value should be changed only if it is known that this will not cause an application to operate incorrectly.

You cannot set a value that is greater than the queue manager's MaxMsgLength attribute.

The lower limit for this parameter is 0. The upper limit depends on the environment. For WebSphere MQ for z/VSE, the maximum is 4 MB.

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue. Note that other factors may cause the queue to be treated as full; for example, it will appear to be full if there is no storage available for a message.

You cannot set a value that is greater than the queue manager's MaxQDepth attribute.

Specify a value in the range 0 through 640 000.

MaxQTriggers (MQCFIN)
Maximum number of concurrent trigger instances for a particular queue (parameter identifier: MQIA_MAX_Q_TRIGGERS).

The maximum number of trigger threads that can be activated simultaneously. This parameter applies to queues with a TriggerType of MQTT_EVERY.

Specify a value in the range 1 through 9999.

MaxQUsers (MQCFIN)
Maximum number of active opens to any particular queue (parameter identifier: MQIA_Q_USERS).

The maximum number of active opens requests against a queue.

You cannot set a value that is greater than the queue manager's MaxQUsers attribute.

Specify a value in the range 1 through 32000.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. This parameter is applicable to Local and Model queues.

The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible by means of the message handle.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the QDepthHighLimit parameter.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application has put a message to a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the QDepthHighEvent parameter.

The value is expressed as a percentage of the maximum queue depth (MaxQDepth attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the QDepthLowLimit parameter.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the QDepthLowEvent parameter.

The value is expressed as a percentage of the maximum queue depth (MaxQDepth attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

A Queue Full event indicates that an MQPUT call to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object.

The maximum length of the string is MQ_Q_DESC_LENGTH.

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

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the QServiceIntervalEvent parameter.

The value is in units of milliseconds, and must be greater than or equal to zero, and less than or equal to 99 999 999.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

A Queue Service Interval High event is generated when a check indicates that no messages have been retrieved from or put to the queue for at least the time indicated by the QServiceInterval attribute.

A Queue Service Interval OK event is generated when a check indicates that a message has been retrieved from the queue within the time indicated by the QServiceInterval attribute.

Note: The value of this attribute can change implicitly. See "Performance events" on page 95.

The value can be:

- MQQSIE_HIGH Queue Service Interval High events enabled.
  - Queue Service Interval High events are enabled, and
  - Queue Service Interval OK events are disabled.
- MQQSIE_OK Queue Service Interval OK events enabled.
  - Queue Service Interval High events are disabled, and
  - Queue Service Interval OK events are enabled.
- MQQSIE_NONE No queue service interval events enabled.
  - Queue Service Interval High events are disabled, and
  - Queue Service Interval OK events are also disabled.

QueueAccounting (MQCFIN)
Controls the collection of accounting data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the QueueAccounting parameter on the queue manager.

MQMON_OFF
Accounting data collection is disabled for the queue.

MQMON_ON
If the value of the queue manager's QueueAccounting parameter is not MQMON_NONE, accounting data collection is enabled for the queue.

QueueStatistics (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_STATISTICS_Q). Specifies whether statistics data collection is enabled.

The value can be:
MQMON_Q_MGR
The value of the queue manager's QueueStatistics parameter is inherited by the queue.
MQMON_OFF
Statistics data collection is disabled
MQMON_ON
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, statistics data collection is enabled.

QueueMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected.

The value can be:
MQMON_OFF
Online monitoring data collection is turned off for this queue.
MQMON_Q_MGR
The value of the queue manager's QueueMonitoring parameter is inherited by the queue. This is the default value.
MQMON_LOW
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this queue.
MQMON_MEDIUM
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this queue.
MQMON_HIGH
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this queue.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, RemoteQName must not be blank when the open occurs.

If this definition is used for a queue-manager alias definition, RemoteQName must be blank when the open occurs.

If this definition is used for a reply-to alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, RemoteQMgrName must not be blank or the name of the connected queue manager. If XmitQName is blank there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a queue-manager alias, RemoteQMgrName is the name of the queue manager, which can be the name of the connected queue manager. Otherwise, if XmitQName is blank, when the queue is opened there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Reorganization (MQCFIN)
Automatic VSAM reorganization (parameter identifier: MQIA_AUTO_REORGANIZATION).

Indicates whether the VSAM file hosting a queue should be scheduled for automatic VSAM reorganization.

The value can be:

MQREORG_ENABLED
Automatic reorganization enabled.

MQREORG_DISABLED
Automatic reorganization disabled.

ReorgStartTime (MQCFST)
Automatic VSAM reorganization start time (parameter identifier: MQCA_AUTO_REORG_START_TIME).

Indicates the time of day, in HHMM format, for the automatic VSAM reorganization to occur following a system restart.

The maximum length of the string is MQ_AUTO_REORG_TIME_LENGTH.

ReorgInterval (MQCFIN)
Automatic VSAM reorganization interval (parameter identifier: MQIA_AUTO_REORG_INTERVAL).

Indicates the frequency, in minutes, for the automatic VSAM reorganization to occur, after its initial activation.

Specify a value between 0 and 1440.

ReorgCatalog (MQCFST)
Automatic VSAM reorganization catalog (parameter identifier: MQCA_AUTO_REORG_CATALOG).

The maximum length of the string is MQ_AUTO_REORG_CATALOG_LENGTH.

The contents of this field are now treated as comments and may not reflect the actual VSAM catalog containing the reorganization file. For the reorganization process, the VSAM catalog where the reorganization file is defined is now extracted from the system and so no longer needs to be specified in the queue definition.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications, can open this queue for input.

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

TriggerChannelName (MQCFST)
Channel name for MCA trigger process (parameter identifier: MQCA_TRIGGER_CHANNEL_NAME).

This parameter is only valid for queues with Usage of MQUS_TRANSMISSION.

For a transmission queue definition, this parameter must identify a channel name.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether message activity on a queue will cause a trigger transaction or program to be started.

The value can be:

MQTC_OFF
Trigger activation not required.

MQTC_ON
Trigger activation required.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger data structure that is passed to a triggered transaction or program.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

TriggerProgramName (MQCFST)
Program name for trigger process (parameter identifier: MQCA_TRIGGER_PROGRAM_NAME).

Specifies the program name that is to be started when a trigger event occurs on the queue.

The maximum length of the string is MQ_TRIGGER_PROGRAM_NAME_LENGTH.

TriggerRestart (MQCFIN)
Indicator for the reactivation of a trigger process (parameter identifier: MQIA_TRIGGER_RESTART).

Specifies whether or not a trigger instance can be restarted when it is detected that there are messages on a queue, but no trigger instance already running.

The value can be:

MQTRIGGER_RESTART_NO
Trigger reactivation not required.

MQTRIGGER_RESTART_YES
Trigger reactivation required.

TriggerTerminalId (MQCFST)
Terminal identifier for trigger process (parameter identifier: MQCA_TRIGGER_TERM_ID).

Specifies a CICS terminal identifier to be associated with a trigger transaction or program instance.

The maximum length of the string is MQ_TRIGGER_TERM_ID_LENGTH.

TriggerTransactionId (MQCFST)
Transaction identifier for trigger process (parameter identifier: MQCA_TRIGGER_TRANS_ID).

Specifies a transaction identifier is to be started when a trigger event occurs on the queue.

The maximum length of the string is MQ_TRIGGER_TRANS_ID_LENGTH.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger transaction or program is started.

The value can be:

MQTT_NONE
No trigger activation.

MQTT_EVERY
Trigger activation for every message.

MQTT_FIRST
Trigger activation when queue depth goes from 0 to 1.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote message queue manager.

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for either a remote queue or for a queue-manager alias definition.

If XmitQName is blank, a queue with the same name as RemoteQMgrName is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue-manager alias and RemoteQMgrName is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Note that the optional parameters described in this section are also applicable to these PCF commands:

- "Copy Queue" on page 290
- "Create Queue" on page 297

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Change Queue Manager

The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

This PCF is supported on all platforms.

For any optional parameters that are omitted, the value does not change.

Required parameters:

None

Optional parameters:

AccountingConnOverride, AccountingInterval, AdoptMCA, AdoptMCACheck, AuthorityEvent, BatchInterfaceAutoStart, BatchInterfaceId, ChannelAuthenticationRecords, ChannelAutoDef, ChannelAutoDefEvent, ChannelAutoDefExit, ChannelEvent, ChannelMonitoring, ChannelStatistics, CodedCharSetId, CommandEvent, CommandInputQName, CommandReplyQName, CommandServerAutoStart, CommandServerDataConversion, CommandServerDeadLetterQ, ConfigurationEvent, ConsCommsMsgs, ConsCritMsgs, ConsErrMsgs, ConsInfoMsgs, ConsReorgMsgs, ConsSystemMsgs, ConsWarnMsgs, CSMTError, DeadLetterQName, InhibitEvent, InternalDump, ListenerPortNumber, LocalEvent, LogCommsMsgs, LogCritMsgs, LogErrMsgs, LogInfoMsgs, LogReorgMsgs, LogSystemMsgs, LogWarnMsgs, MaxClients, MaxGlobalLocks, MaxHandles, MaxLocalLocks, MaxMsgLength, MaxOpenQ, MaxPropertiesLength, MaxQDepth, MaxQUsers, MonitorInterval, MonitorQName, MQIAccounting, MQIStatistics, PerformanceEvent, QMgrDesc, QueueAccounting, QueueMonitoring, QueueStatistics, RecoveryTasks, RemoteEvent, SSLEvent, SSLKeyLibraryMember, SSLKeyLibraryName, SSLKeyResetCount, StartStopEvent, StatisticsInterval, SystemLogQName, TraceComms, TraceConversion, TraceMQICalls, TraceReorg, TraceSystem

Optional parameters

AccountingConnOverride (MQCFIN)
Specifies whether applications can override the settings of the QueueAccounting and MQIAccounting queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be:

MQMON_DISABLED
Applications cannot override the settings of the QueueAccounting and MQIAccounting parameters. This is the queue manager's initial default value.

MQMON_ENABLED
Applications can override the settings of the QueueAccounting and

MQIAccounting parameters by using the options field of the MQCNO structure of the MQCONNX API call.

AccountingInterval (MQCFIN)
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

Specify a value in the range 1 through 604 000.

AdoptMCA (MQCFIN)
Adopt MCA (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

Controls whether the channel Adopt MCA feature is active for the queue manager.

The value can be:

MQADOPT_TYPE_NO
Adopt MCA feature is disabled.

MQADOPT_TYPE_RCVR
Adopt MCA feature is enabled for receiver channels.

AdoptMCACheck (MQCFIN)
Adopt MCA check (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

Controls whether the Adopt MCA feature checks the partner net address when adopting an MCA instance.

The value can be:

MQADOPT_CHECK_NONE
The net address is not checked when an MCA instance is adopted.

MQADOPT_CHECK_NET_ADDR
The net address is checked when an MCA instance is adopted.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

BatchInterfaceAutoStart (MQCFIN)
Indicator for the automatic activation of the batch interface (parameter identifier: MQIA_BATCH_INTERFACE_AUTO).

Specifies whether the WebSphere MQ for z/VSE batch interface is automatically started when the local queue manager is started.

The value can be:

MQAUTO_START_NO
Do not auto-start the batch interface.

MQAUTO_START_YES
Auto-start the batch interface.

BatchInterfaceId (MQCFST)
Batch interface identifier (parameter identifier: MQCA_BATCH_INTERFACE_ID).

Specifies a unique batch interface identifier. Batch programs should specify a SETPARM card in their JCL that identifies the appropriate batch interface identifier. For z/VSE systems running multiple queue managers, the identifier must be unique.

The maximum length of the string is MQ_BATCH_INTERFACE_ID_LENGTH.

ChannelAuthenticationRecords(MQCFIN)
Controls whether channel authentication records are used. Channel attribute (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:
MQCHLA_DISABLED
Channel authentication records are not checked.
MQCHLA_ENABLED
Channel authentication records are checked.

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

The value can be:

MCCHAD_DISABLED
Channel auto-definition disabled

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver or server-connection channel is auto-defined.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: QCA_CHANNEL_AUTO_DEF_EXIT).

This exit is invoked when an inbound request for an undefined channel is received. The channel auto-definition is enabled (see ChannelAutoDef).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ChannelEvent (MQCFIN)
Channel event (parameter identifier: MQIA_CHANNEL_EVENT).

Controls whether channel-related events are generated.

The value can be:

MQEVR_ENABLED
Channel events enabled.

MQEVR_DISABLED
Channel events disabled.

Note: Enabling channel events enables all channel event types. Disabling channel events disables all channel events. Individual channel events can be set using the WebSphere MQ for z/VSE master terminal transactions.

ChannelMonitoring (MQCFIN)
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their ChannelMonitoring parameter.

MQMON_OFF
Online monitoring data collection is turned off for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter. This is the queue manager's initial default value.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.

ChannelStatistics (MQCFIN)
Controls whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:
MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their ChannelStatistics parameter. This is the queue manager's initial default value.
MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.
MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.

CodedCharSetId (MQCFIN)
Queue manager coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier (CCSID) for the queue manager. The CCSID is the identifier used with all character string fields defined by the application programming interface (API). It does not apply to application data carried in the text of a message unless the CCSID in the message descriptor, when the message is put with an MQPUT or MQPUT1, is set to the value MQCCSI_Q_MGR.

Specify a value in the range 1 through 65 535.

The CCSID must specify a value that is defined for use on the platform and use an appropriate character set. For a list of supported CCSIDs for WebSphere MQ for z/VSE, review your LE/VSE configuration.

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT). The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandInputQName (MQCFST)
PCF command input queue (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

Specifies the name of the PCF command input queue. The PCF command input queue expects PCF messages from local or remote administration applications, and is monitored by the WebSphere MQ Command Server. If this parameter is changed, you are advised to stop and restart the command server so that a new queue name is recognized.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandReplyQName (MQCFST)
WebSphere MQ command reply queue (parameter identifier: MQCA_COMMAND_REPLY_Q_NAME).

Specifies the name of the WebSphere MQ command reply queue. The WebSphere MQ command reply queue is used by the WebSphere MQ command utility for responses to WebSphere MQ commands issued from a batch partition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandServerAutoStart (MQCFIN)
Indicator for the automatic activation of the PCF command server (parameter identifier: MQIA_CMD_SERVER_AUTO).

Specifies whether the WebSphere MQ for z/VSE PCF command server is automatically started when the local queue manager is started.

The value can be:

MQAUTO_START_NO
Do not auto-start the PCF command server.

MQAUTO_START_YES
Auto-start the PCF command server.

CommandServerDataConversion (MQCFIN)
Indicator for the data conversion of PCF messages (parameter identifier: MQIA_CMD_SERVER_CONVERT_MSG).

Specifies whether the WebSphere MQ for z/VSE PCF command server is to apply data conversion to PCF messages read from the PCF command input queue.

The value can be:

MQCSRV_CONVERT_NO
Do not convert PCF messages.

MQCSRV_CONVERT_YES
Convert PCF messages.

CommandServerDeadLetterQ (MQCFST)
Indicator for the storage of undeliverable PCF reply messages to the system dead letter queue(parameter identifier: MQIA_CMD_SERVER_DLQ_MSG).

Specifies whether the WebSphere MQ for z/VSE PCF command server is to place undeliverable PCF command responses to the system dead letter queue.

The value can be:

MQCSRV_DLQ_NO
Do not put undeliverable PCF responses to the system dead letter queue.

MQCSRV_DLQ_YES
Put undeliverable PCF responses to the system dead letter queue.

ConfigurationEvent (MQCFIN)
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT). The value can be:
MQEVR_DISABLED
Event reporting disabled.
MQEVR_ENABLED
Event reporting enabled.

ConsCommsMsgs (MQCFIN)
Log communications messages to console (parameter identifier: MQIA_QMOPT_CONS_COMMS_MSGS).

Indicates whether the queue manager sends messages generated by the communications subsystem to the console.

The value can be:

MQQMOPT_ENABLED
Communications messages sent to the console.

MQQMOPT_DISABLED
Communications messages not sent to the console.

MQQMOPT_REPLY
Communications messages sent to the console, and the operator prompted to reply.

ConsCritMsgs (MQCFIN)
Log critical messages to console (parameter identifier: MQIA_QMOPT_CONS_CRITICAL_MSGS).

Indicates whether the queue manager sends messages of critical severity to the console.

The value can be:

MQQMOPT_ENABLED
Critical messages sent to the console.

MQQMOPT_DISABLED
Critical messages not sent to the console.

MQQMOPT_REPLY
Critical messages sent to the console, and the operator prompted to reply.

ConsErrMsgs (MQCFIN)
Log error messages to console (parameter identifier: MQIA_QMOPT_CONS_ERROR_MSGS).

Indicates whether the queue manager sends messages of error severity to the console.

The value can be:

MQQMOPT_ENABLED
Error messages sent to the console.

MQQMOPT_DISABLED
Error messages not sent to the console.

MQQMOPT_REPLY
Error messages sent to the console, and the operator prompted to reply.

ConsInfoMsgs (MQCFIN)
Log informational messages to console (parameter identifier: MQIA_QMOPT_CONS_INFO_MSGS).

Indicates whether the queue manager sends messages of informational severity to the console.

The value can be:

MQQMOPT_ENABLED
Informational messages sent to the console.

MQQMOPT_DISABLED
Informational messages not sent to the console.

ConsReorgMsgs (MQCFIN)
Log reorganization messages to console (parameter identifier: MQIA_QMOPT_CONS_REORG_MSGS).

Indicates whether the queue manager sends messages generated by the reorganization subsystem to the console.

The value can be:

MQQMOPT_ENABLED
Reorganization messages sent to the console.

MQQMOPT_DISABLED
Reorganization messages not sent to the console.

MQQMOPT_REPLY
Reorganization messages sent to the console, and the operator prompted to reply.

ConsSystemMsgs (MQCFIN)
Log system messages to console (parameter identifier: MQIA_QMOPT_CONS_SYSTEM_MSGS).

Indicates whether the queue manager sends general operational messages to the console.

The value can be:

MQQMOPT_ENABLED
System messages sent to the console.

MQQMOPT_DISABLED
System messages not sent to the console.

MQQMOPT_REPLY
System messages sent to the console, and the operator prompted to reply.

ConsWarnMsgs (MQCFIN)
Log warning messages to console (parameter identifier: MQIA_QMOPT_CONS_WARNING_MSGS).

Indicates whether the queue manager sends messages of warning severity to the console.

The value can be:

MQQMOPT_ENABLED
Warning messages sent to the console.

MQQMOPT_DISABLED
Warning messages not sent to the console.

CSMTError (MQCFIN)
Allow TDQ Write on Error (parameter identifier: MQIA_QMOPT_CSMT_ON_ERROR).

Indicates whether operational messages are sent to the CICS CSMT when the system log queue is unavailable.

The value can be:

MQQMOPT_ENABLED
Operational messages sent to the CSMT TDQ when the system log queue is unavailable.

MQQMOPT_DISABLED
Operational messages not sent to the CSMT TDQ when the system log queue is unavailable.

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

InternalDump (MQCFIN)
Allow internal CICS dump (parameter identifier: MQIA_QMOPT_INTERNAL_DUMP).

Indicates whether the queue manager generates a CICS dump when an MQI application generates an unrecoverable error.

The value can be:

MQQMOPT_ENABLED
Application internal dump enabled.

MQQMOPT_DISABLED
Application internal dump disabled.

ListenerPortNumber (MQCFIN)
Port number for TCP/IP Listener process (parameter identifier: MQIA_LISTENER_PORT_NUMBER).

Specifies the TCP/IP port number that WebSphere MQ uses for accepting TCP/IP connection requests from remote queue managers and MQ clients. The default port value is 1414, however any unreserved port number can be used.

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

LogCommsMsgs (MQCFIN)
Log communications messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_COMMS_MSGS).

Indicates whether the queue manager sends messages generated by the communications subsystem to the system log.

The value can be:

MQQMOPT_ENABLED
Communications messages sent to the system log.

MQQMOPT_DISABLED
Communications messages not sent to the system log.

LogCritMsgs (MQCFIN)
Log critical messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_CRITICAL_MSGS).

Indicates whether the queue manager sends messages of critical severity to the system log queue.

The value can be:

MQQMOPT_ENABLED
Critical messages sent to the system log.

MQQMOPT_DISABLED
Critical messages not sent to the system log.

LogErrMsgs (MQCFIN)
Log error messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_ERROR_MSGS).

Indicates whether the queue manager sends messages of error severity to the system log queue.

The value can be:

MQQMOPT_ENABLED
Error messages sent to the system log.

MQQMOPT_DISABLED
Error messages not sent to the system log.

LogInfoMsgs (MQCFIN)
Log informational messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_INFO_MSGS).

Indicates whether the queue manager sends messages of informational severity to the system log queue.

The value can be:

MQQMOPT_ENABLED
Informational messages sent to the system log.

MQQMOPT_DISABLED
Informational messages not sent to the system log.

LogReorgMsgs (MQCFIN)
Log reorganization messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_REORG_MSGS).

Indicates whether the queue manager sends messages generated by the reorganization subsystem to the system log.

The value can be:

MQQMOPT_ENABLED
Reorganization messages sent to the system log.

MQQMOPT_DISABLED
Reorganization messages not sent to the system log.

LogSystemMsgs (MQCFIN)
Log system messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_SYSTEM_MSGS).

Indicates whether the queue manager sends general operational messages to the system log.

The value can be:

MQQMOPT_ENABLED
System messages sent to the system log.

MQQMOPT_DISABLED
System messages not sent to the system log.

LogWarnMsgs (MQCFIN)
Log warning messages to the system log queue (parameter identifier: MQIA_QMOPT_LOG_WARNING_MSGS).

Indicates whether the queue manager sends messages of warning severity to the system log queue.

The value can be:

MQQMOPT_ENABLED
Warning messages sent to the system log.

MQQMOPT_DISABLED
Warning messages not sent to the system log.

MaxClients (MQCFIN)
Number of licensed clients (parameter identifier: MQIA_MAX_CLIENTS).

Specifies the maximum number of licensed clients that can establish a server connection at any one time.

Specify a value between 0 and 99999.

MaxGlobalLocks (MQCFIN)
Buffer size for queue manager to manage concurrent queue access (parameter identifier: MQIA_MAX_GLOBAL_LOCKS).

The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls, for each queue in the system, for recovery.

A value of 500 is normally sufficient.

The maximum value is 1000.

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

The maximum number of connection handles that the queue manager will manage at any one time.

Specify a value in the range 1 through 1000.

MaxLocalLocks (MQCFIN)
Buffer size for applications to manage concurrent queue access (parameter identifier: MQIA_MAX_LOCAL_LOCKS).

The maximum number of entries that the queue manager can use to maintain uncommitted MQPUT or MQGET calls for each queue and task for recovery.

A value of 500 is normally sufficient.

The maximum value is 1000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length of messages allowed on queues on the queue manager. Changing this parameter does not affect existing queue definitions.

If you reduce the maximum message length for the queue manager, you should verify that existing queues do not already exceed the new MaxMsgLength value.

The lower limit for this parameter is 0. The upper limit for WebSphere MQ for z/VSE is 4 MB.

MaxPropertiesLength (MQCFIN)
Maximum property length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

Specifies the maximum length of the properties, including both the property name in bytes and the size of the property value in bytes.

Specify a value in the range 0 through 4 MB (4194304 bytes), or the special value, MQPROP_UNRESTRICTED_LENGTH:

MQPROP_UNRESTRICTED_LENGTH
The size of the properties is restricted only by the upper limit.

MaxQOpen (MQCFIN)
Maximum number of concurrently open queues (parameter identifier: MQIA_MAX_OPEN_Q).

Specifies the maximum number of queues any single task can have open at any one time.

Specify a value in the range 1 through 1000.

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on a queue. Note that other factors may cause the queue to be treated as full; for example, it will appear to be full if there is no storage available for a message.

If you reduce the maximum queue depth for the queue manager, you should verify that existing queues do not already exceed the new MaxQDepth value.

Specify a value in the range 0 through 640 000.

MaxQUsers (MQCFIN)
Maximum number of active opens to any particular queue (parameter identifier: MQIA_Q_USERS).

The maximum number of open requests that the queue manager will manage for a single queue.

Specify a value in the range 1 through 32000.

MonitorInterval (MQCFIN)
Queue manager housekeeping process interval (parameter identifier: MQIA_MONITOR_INTERVAL).

Specifies the interval (in seconds) that the queue manager housekeeping task suspends during process iterations.

A value of 30 seconds is usually sufficient.

MonitorQName (MQCFST)
MQI monitor queue name (parameter identifier: MQCA_MONITOR_Q_NAME).

Specifies the name of the local queue that is to be used for MQI diagnostic messages. The MQI Monitor when active, places diagnostic messages to the monitor queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

MQIAccounting (MQCFIN)
Controls whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:
MQMON_OFF
MQI accounting data collection is disabled. This is the queue manager's initial default value.
MQMON_ON
MQI accounting data collection is enabled.

MQIStatistics (MQCFIN)
Controls whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:
MQMON_OFF
Data collection for MQI statistics is disabled. This is the queue manager's initial default value.
MQMON_ON
Data collection for MQI statistics is enabled.

PerformanceEvent (MQCFIN))
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QMgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing, to ensure that the text is translated correctly.

QueueAccounting (MQCFIN)
Controls the collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_NONE
Accounting data collection for queues is disabled. This may not be overridden by the value of the QueueAccounting parameter on the queue.

MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the QueueAccounting parameter.

MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the QueueAccounting parameter.

QueueMonitoring (MQCFIN)
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the QueueMonitoring queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off. This is the queue manager's initial default value.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their QueueMonitoring attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

QueueStatistics (MQCFIN)
Controls whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their QueueStatistics parameter. This is the queue manager's initial default value.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their QueueStatistics parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their QueueStatistics parameter.

RecoveryTasks (MQCFIN)
Maximum recovery tasks (parameter identifier: MQIA_MAX_RECOVERY_TASKS).

Indicates the maximum number of CICS tasks that the queue manager will start to resolve discrepancies in dual queues.

Specify a value between 0 and 9999.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLEvent (MQCFIN)
Controls whether SSL events are generated (parameter identifier: MQIA_SSL_EVENT). The value can be:
MQEVR_DISABLED
Event reporting disabled.
MQEVR_ENABLED
Event reporting enabled.

SSLKeyLibraryMember (MQCFST)
SSL key library name (parameter identifier: MQCA_SSL_KEY_LIBRARY).

Specifies the SSL key-ring sublibrary. The key-ring sublibrary contains private key and X.509 certificate files.

Specify a valid z/VSE sublibrary name.

The maximum length of the string is MQ_SSL_KEY_LIBRARY_LENGTH.

SSLKeyLibraryName (MQCFST)
SSL key member name (parameter identifier: MQCA_SSL_KEY_MEMBER).

Specifies the SSL key-ring member name of the private key and certificate files that will be used by WebSphere MQ enabled channels.

This must be a valid z/VSE sublibrary member name.

It should be noted that WebSphere MQ for z/VSE uses the same private key and certificate for all SSL enabled channels. It is not possible to identify a different certificate on a per channel basis. Consequently, the key-ring member name should identify a private key and certificate files appropriate for all SSL enabled channels.

The maximum length of the string is MQ_SSL_KEY_MEMBER_LENGTH.

SSLKeyResetCount (MQCFIN)
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

Specifies when SSL channel MCAs that initiate communication reset the secret key used for encryption on the channel. The value of this parameter represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. This number of bytes includes control information sent by the MCA. The secret key is renegotiated when (whichever occurs first):

- The total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or
- If channel heartbeats are enabled, before data is sent or received following a channel heartbeat.

Specify a value in the range zero through 999 999 999. A value of zero, the queue manager's initial default value, signifies that secret keys are never renegotiated.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StatisticsInterval (MQCFIN)
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

Specify a value in the range 1 through 604 000.

SystemLogQName (MQCFST)
System log queue name (parameter identifier: MQCA_SYSTEM_LOG_Q_NAME).

Specifies the name of the system log queue that is used by WebSphere MQ for z/VSE to store operational diagnostic and error messages.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TraceComms (MQCFIN)
Trace communication events (parameter identifier: MQIA_QMOPT_TRACE_COMMS).

Indicates whether the queue manager traces communication diagnostics to the CICS auxiliary trace.

The value can be:

MQQMOPT_ENABLED
Tracing of communication events enabled.

MQQMOPT_DISABLED
Tracing of communication events disabled.

TraceConversion (MQCFIN)
Trace data conversion events (parameter identifier: MQIA_QMOPT_TRACE_CONVERSION).

Indicates whether the queue manager traces data conversion diagnostics to the CICS auxiliary trace.

The value can be:

MQQMOPT_ENABLED
Tracing of data conversion events enabled.

MQQMOPT_DISABLED
Tracing of data conversion events disabled.

TraceMQICalls (MQCFIN)
Trace MQI call events (parameter identifier: MQIA_QMOPT_TRACE_MQI_CALLS).

Indicates whether the queue manager traces MQI call diagnostics to the CICS auxiliary trace.

The value can be:

MQQMOPT_ENABLED
Tracing of MQI calls enabled.

MQQMOPT_DISABLED
Tracing of MQI calls disabled.

TraceReorg (MQCFIN)
Trace reorganization events (parameter identifier: MQIA_QMOPT_TRACE_REORG).

Indicates whether the queue manager traces reorganization diagnostics to the CICS auxiliary trace.

The value can be:

MQQMOPT_ENABLED
Tracing of reorganization events enabled.

MQQMOPT_DISABLED
Tracing of reorganization events disabled.

TraceSystem (MQCFIN)
Trace system events (parameter identifier: MQIA_QMOPT_TRACE_SYSTEM).

Indicates whether the queue manager traces general system diagnostics to the CICS auxiliary trace.

The value can be:

MQQMOPT_ENABLED
Tracing of system events enabled.

MQQMOPT_DISABLED
Tracing of system events disabled.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_MGR_CCSID_ERROR
Coded character set value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_UNKNOWN_Q_MGR
Queue manager not known.

Change Service

The Change Service (MQCMD_CHANGE_SERVICE) command changes the specified attributes of an existing WebSphere MQ service definition.

For any optional parameters that are omitted, the value does not change.

Required parameters:

ServiceName

Optional parameters:

ServiceDesc, ServiceType, StartArguments, StartCommand, StartMode, StopArguments, StopCommand

Required parameters

For details of the required parameters for Change Service, see “Create Service” on page 299.

Optional parameters

For details of the optional parameters for Change Service, see “Create Service” on page 299.

Copy Channel

The Copy Channel (MQCMD_COPY_CHANNEL) command creates a new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

This PCF is supported on all platforms.

Required parameters:

FromChannelName, ToChannelName, ChannelType

Optional parameters:

BatchInterval, BatchSize, ChannelDesc, ChannelStatistics, ChannelMonitoring, ConnectionName, DataConversion, DiscInterval, DiscRetryCount, LongRetryCount, LongRetryInterval, MaxMsgLength, MsgExit, MsgUserData, PortNumber, PropertyControl, ReceiveExit, ReceiveUserData, SecurityExit, SecurityUserData, SendExit, SendUserData, SeqNumberWrap, ShortRetryCount, ShortRetryInterval, SSLCipherSpec, SSLClientAuth, SSLPeerName, TpName, TransportType, XmitQName

Required parameters

FromChannelName (MQCFST)

From channel name (parameter identifier: MQCACF_FROM_CHANNEL_NAME).

The name of the existing channel definition that contains values for the attributes that are not specified in this command.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ToChannelName (MQCFST)

To channel name (parameter identifier: MQCACF_TO_CHANNEL_NAME).

The name of the new channel definition.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name already exists, the command will fail.

ChannelType (MQCFIN)

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being copied. The value can be:

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_SVRCONN
Server-connection (for use by clients).

Optional parameters

For a complete list and description of the optional parameters available with the Copy Channel command, refer to “Change Channel” on page 244.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_ALREADY_EXISTS
Channel already exists.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONN_NAME_ERROR
Error in connection name parameter.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_LONG_RETRY_ERROR
Long retry value is not valid.

MQRCCF_LONG_TIMER_ERROR
Long retry interval is not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry value is not valid.

MQRCCF_SHORT_TIMER_ERROR
Short retry interval is not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Copy Channel Listener

The Copy Channel Listener (MQCMD_COPY_LISTENER) command creates a new WebSphere MQ listener definition, using, for attributes not specified in the command, the attribute values of an existing listener definition.

Required parameters:

FromListenerName, ToListenerName

Optional parameters:

Backlog, IPAddress, ListenerDesc, Port, StartMode, TransportType

Required parameters

FromListenerName (MQCFST)
The name of the listener definition to be copied from (parameter identifier: MQCACF_FROM_LISTENER_NAME).

This specifies the name of the existing listener definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

ToListenerName (MQCFST)
To listener name (parameter identifier: MQCACF_TO_LISTENER_NAME).

This specifies the name of the new listener definition.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

For details of the optional parameters for Copy Channel Listener, see "Create Channel Listener" on page 295.

Copy Namelist

The Copy Namelist (MQCMD_COPY_NAMELIST) command copies an existing namelist definition and uses it to create a new namelist.

For any optional parameters that are omitted, the value of the existing namelist definition is used.

Required parameters

FromNamelistName, To NamelistName

Optional parameters

NamelistDesc, Names

Required parameters

FromNamelistName (MQCFST)
The name of the namelist definition to be copied from (parameter identifier: MQCACF_FROM_NAMELIST_NAME).

This specifies the name of the existing namelist definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

ToNamelistName (MQCFST)
To namelist name (parameter identifier: MQCACF_TO_NAMELIST_NAME).

This specifies tha name of the new namelist definition.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This is a plain-text comment that provides descriptive information about the namelist definition. It should contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

Names (MQCFSL)
The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the Count field in the MQCFSL structure. The length of each name is given by the StringLength field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Copy Queue

The Copy Queue (MQCMD_COPY_Q) command creates a new queue definition, of the same type, using, for attributes not specified in the command, the attribute values of an existing queue definition.

This PCF is supported on all platforms.

Required parameters:

FromQName, ToQName, QType

Optional parameters:

BaseQName, CICSFileName, DefinitionType, InhibitGet, InhibitPut, MaxGlobalLocks, MaxLocalLocks, MaxMsgLength, MaxQDepth, MaxQTriggers, MaxQUsers, PropertyControl, QDepthHighEvent, QDepthHighLimit, QDepthLowEvent, QDepthLowLimit, QDepthMaxEvent, QDesc, QServiceInterval, QServiceIntervalEvent, QueueAccounting, QueueStatistics, QueueMonitoring, RemoteQMgrName, RemoteQName, Shareability, TriggerChannelName, TriggerControl, TriggerData, TriggerProgramName, TriggerRestart, TriggerTerminalId, TriggerTransactionId, TriggerType, Usage, XmitQName

Required parameters

FromQName (MQCFST)

From queue name (parameter identifier: MQCACF_FROM_Q_NAME).

Specifies the name of the existing queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ToQName (MQCFST)

To queue name (parameter identifier: MQCACF_TO_Q_NAME).

Specifies the name of the new queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue names must be unique; if a queue definition exists with the same name as the new queue, the command will fail.

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being copied.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Optional parameters

For a complete list and description of the optional parameters available with the Copy Queue command, refer to "Change Queue" on page 257.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_LIKE_OBJECT_WRONG_TYPE
New and existing objects have different type.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_OBJECT_WRONG_TYPE
Object has wrong type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Copy Service

The Copy Service (MQCMD_COPY_SERVICE) command creates a new WebSphere MQ service definition, using, for attributes not specified in the command, the attribute values of an existing service definition.

Required parameters:

FromServiceName, ToServiceName

Optional parameters:

ServiceDesc, ServiceType, StartArguments, StartCommand, StartMode, StopArguments, StopCommand

Required parameters

FromServiceName (MQCFST)
The name of the service definition to be copied from (parameter identifier: MQCACF_FROM_SERVICE_NAME).

This specifies the name of the existing service definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ToServiceName (MQCFST)
To service name (parameter identifier: MQCACF_TO_SERVICE_NAME).

This specifies the name of the new service definition.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

For details of the optional parameters for Copy Service, see “Create Service” on page 299.

Create Channel

The Create Channel (MQCMD_CREATE_CHANNEL) command creates an WebSphere MQ channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

This PCF is supported on all platforms.

Required parameters:

ChannelName, ChannelType

Optional parameters:

BatchInterval, BatchSize, ChannelDesc, ChannelStatistics, ChannelMonitoring, ConnectionName, DataConversion, DiscInterval, DiscRetryCount,

LongRetryCount, LongRetryInterval, MaxMsgLength, MsgExit, MsgUserData, PortNumber, PropertyControl, ReceiveExit, ReceiveUserData, SecurityExit, SecurityUserData, SendExit, SendUserData, SeqNumberWrap, ShortRetryCount, ShortRetryInterval, SSLCipherSpec, SSLClientAuth, SSLPeerName, TpName, TransportType, XmitQName

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the new channel definition. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name already exists, the command will fail.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being defined. The value can be:

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_SVRCONN
Server-connection (for use by clients).

Optional parameters

For a complete list and description of the optional parameters available with the Copy Channel command, refer to “Change Channel” on page 244.

Error codes

In addition to the values for any command shown in section “Error codes applicable to all commands” on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_ALREADY_EXISTS
Channel already exists.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONN_NAME_ERROR
Error in connection name parameter.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_LONG_RETRY_ERROR
Long retry value is not valid.

MQRCCF_LONG_TIMER_ERROR
Long retry interval is not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry value is not valid.

MQRCCF_SHORT_TIMER_ERROR
Short retry interval is not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Create Channel Listener

The Create Channel Listener (MQCMD_CREATE_LISTENER) command creates a new WebSphere MQ listener definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters:

ListenerName, TransportType

Optional parameters:

Backlog, IPAddress, ListenerDesc, Port, StartMode

Required parameters

ListenerName (MQCFST)
The name of the listener definition to be changed or created (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be: MQXPT_TCP TCP

Optional parameters

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured Ipv4 stacks.

The maximum length of the string is MQ_CONN_ADDRESS_LENGTH.

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

This is a plain-text comment that provides descriptive information about the listener definition. It should contain only displayable characters. If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT). The port number for TCP/IP.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. This is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener 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.

Create Namelist

The Create Namelist (MQCMD_CREATE_NAMELIST) command creates a namelist object using the specified values.

Required parameters

NamelistName

Optional parameters

NamelistDesc, Names

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME). The name of the namelist to be created.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional Parameters

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This is a plain-text comment that provides descriptive information about the namelist definition. It should contain only displayable characters

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

Names (MQCFSL)
The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the Count field in the MQCFSL structure. The length of each name is given by the StringLength field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Create Queue

The Create Queue (MQCMD_CREATE_Q) command creates a queue definition with the specified attributes. All attributes that are not specified are set to the default value for the type of queue that is created.

This PCF is supported on all platforms.

Required parameters:

QName, QType, CICSFileName

Optional parameters:

BaseQName, DefinitionType, InhibitGet, InhibitPut, MaxGlobalLocks, MaxLocalLocks, MaxMsgLength, MaxQDepth, MaxQTriggers, MaxQUsers, PropertyControl, QDepthHighEvent, QDepthHighLimit, QDepthLowEvent, QDepthLowLimit, QDepthMaxEvent, QDesc, QServiceInterval, QServiceIntervalEvent, QueueAccounting, QueueStatistics, QueueMonitoring, RemoteQMgrName, RemoteQName, Shareability, TriggerChannelName, TriggerControl, TriggerData, TriggerProgramName, TriggerRestart, TriggerTerminalId, TriggerTransactionId, TriggerType, Usage, XmitQName

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be created. The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue names must be unique; if a queue definition already exists with the same name as of the new queue, the command will fail.

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_REMOTE
Local definition of a remote queue.

CICSFileName (MQCFST)

CICS file name for queue messages.

The name of a filename defined to the CICS region. The maximum length of the string is MQ_CICS_FILE_NAME_LENGTH.

Optional parameters

For a complete list and description of the optional parameters available with the Create Queue command, refer to "Change Queue" on page 257.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_OBJECT_WRONG_TYPE
Object has wrong type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Create Service

The Create Service (MQCMD_CREATE_SERVICE) command creates a new WebSphere MQ service definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters:

ServiceName

Optional parameters:

ServiceDesc, ServiceType, StartArguments, StartCommand, StartMode, StopArguments, StopCommand

Required parameters

ServiceName (MQCFST)
The name of the service definition to be created (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

This is a plain-text comment that provides descriptive information about the service definition. It should contain only displayable characters. If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceType (MQCFIN)
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE). Specify one of these:

MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.

MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments (MQCFST)
Arguments to be passed in CICS COMMAREA when the CICS transaction is started (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is 100.

StartCommand (MQCFST)
CICS transaction code to start service (parameter identifier: MQCA_SERVICE_START_COMMAND).

The maximum length of the string is 4.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command. This is the default value.

MQSVC_CONTROL_Q_MGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
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.

StopArguments (MQCFST)
Specifies the CICS COMMAREA contents when the CICS transaction is started (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is 100.

StopCommand (MQCFST)
CICS transaction code to stop the service (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

The maximum length of the string is 4.

Delete Channel

The Delete Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

This PCF is supported on all platforms.

Required parameters:
ChannelName

Optional parameters:
None

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Error codes

In addition to the values for any command shown in section “Error codes applicable to all commands” on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Delete Channel Listener

The Delete Channel Listener (MQCMD_DELETE_LISTENER) command deletes an existing channel listener definition.

Required parameters:

ListenerName

Optional parameters:

None

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME). This is the name of the listener definition to be deleted.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Delete Namelist

The Delete Namelist (MQCMD_DELETE_NAMELIST) command deletes a namelist object.

Required parameters:

NamelistName

Optional parameters:

None

Required parameters

NamelistName(MQCFST)
NamelistName (parameter identifier: MQCA_NAMELIST_NAME). The name of the namelist to be deleted.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Delete Queue

The Delete Queue (MQCMD_DELETE_Q) command deletes an WebSphere MQ queue.

This PCF is supported on all platforms.

Required parameters:
QName

Optional parameters:
Purge, QType

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be deleted.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, the queue must be of the specified type.

The value may be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Purge (MQCFIN)
Purge queue (parameter identifier: MQIACF_PURGE).

If there are messages on the queue MQPO_YES must be specified, otherwise the command will fail. If this parameter is not present the queue is not purged.

Valid only for queue of type local.

The value can be:

MQPO_YES
Purge the queue.

MQPO_NO
Do not purge the queue.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PURGE_VALUE_ERROR
Purge value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Delete Service

The Delete Service (MQCMD_DELETE_SERVICE) command deletes an existing service definition.

Required parameters:

ServiceName

Optional parameters:

None

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This is the name of the service definition to be deleted.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Escape

The Escape (MQCMD_ESCAPE) command conveys any WebSphere MQ command to a remote queue manager.

The Escape command can also be used to send a command for which no Programmable Command Format has been defined.

The only type of command that can be carried is one that is identified as an WebSphere MQ command that is recognized at the receiving queue manager.

Required parameters:
EscapeType, EscapeText

Optional parameters:
None

The Escape command, if successful, generates a data response. For details of the Escape response, refer to "Data responses to commands" on page 381.

Required parameters

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
WebSphere MQ command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string to hold a command. The length of the string is limited only by the size of the message. WebSphere MQ for z/VSE supports PCF message lengths up to 2 KB.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_ESCAPE_TYPE_ERROR
Escape type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

Inquire Channel

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of WebSphere MQ channel definitions.

This PCF is supported on all platforms.

Required parameters:

ChannelName

Optional parameters:

ChannelAttrs, ChannelType, IntegerFilterCommand, StringFilterCommand

The Inquire Channel command, if successful, generates a data response. For details of the Inquire Channel response, refer to "Data responses to commands" on page 381.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, the channel specified by ChannelName, must be of the specified type.

If this parameter is not present (or if MQCHT_ALL is specified), the channel identified by ChannelName can be of any type.

The value can be:

MQCHT_ALL
All types.

The default value if this parameter is not specified is MQCHT_ALL. Note: If this parameter is present, it must occur immediately after the ChannelName parameter. Failure to do this can result in a MQRCCF_MSG_LENGTH_ERROR error message.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_SVRCONN
Server-connection (for use by clients).

ChannelAttrs (MQCFIL)
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCACH_CHANNEL_NAME
Channel name.

MQCACH_CONNECTION_NAME
Connection name.

MQCACH_DESC
Description.

MQCACH_MSG_EXIT_NAME
Message exit name.

MQCACH_MSG_EXIT_USER_DATA
Message exit user data.

MQCACH_RCV_EXIT_NAME
Receive exit name.

MQCACH_RCV_EXIT_USER_DATA
Receive exit user data.

MQCACH_SEC_EXIT_NAME
Security exit name.

MQCACH_SEC_EXIT_USER_DATA
Security exit user data.

MQCACH_SEND_EXIT_NAME
Send exit name.

MQCACH_SEND_EXIT_USER_DATA
Send exit user data.

MQCACH_SSL_CIPHER_SPEC
SSL cipher specification.

MQCACH_SSL_PEER_NAME
SSL peer name.

MQCACH_TP_NAME
Transaction program name.

MQCACH_XMIT_Q_NAME
Transmission queue name.

MQIA_MONITORING_CHANNEL
Channel monitoring setting.

MQIA_PROPERTY_CONTROL
Property control attribute.

MQIA_STATISTICS_CHANNEL
Channel statistics setting.

MQIACH_BATCH_INTERVAL
Batch interval.

MQIACH_BATCH_SIZE
Batch size.

MQIACH_CHANNEL_TYPE
Channel type.

MQIACH_DATA_CONVERSION
Whether sender should convert application data.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_DISC_RETRY
Disconnection retry count.

MQIACH_LONG_RETRY
Long retry count.

MQIACH_LONG_TIMER
Long retry interval.

MQIACH_MAX_MSG_LENGTH
Maximum message length.

MQIACH_PORT_NUMBER
TCP/IP port number.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_SHORT_RETRY
Short retry count.

MQIACH_SHORT_TIMER
Short retry interval.

MQIACH_SSL_CLIENT_AUTH
SSL client authentication.

MQIACH_XMIT_PROTOCOL_TYPE
Transport (transmission protocol) type.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Channel Authentication Records

The Inquire Channel Authentication Records (MQCMD_INQUIRE_CHLAUTH_RECS) command retrieves the allowed partner details and mappings to MCAUSER for a channel or set of channels.

Required parameters:

generic-channel-name

Optional parameters:

Address, ChannelAuthAttrs, ClntUser, IntegerFilterCommand, QMName, StringFilterCommand, Type

Required parameters

generic-channel-name
The name of the channel or set of channels on which you are inquiring. You can use the asterisk (*) as a wildcard to specify a set of channels, unless you set Match to MQMATCH_RUNCHECK. If you set Type to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

Optional parameters

Address (MQCFST)
The IP address to be mapped (parameter identifier: MQCACH_CONNECTION_NAME). This parameter is valid only when Match is MQMATCH_RUNCHECK and must not be generic.

ChannelAuthAttrs (MQCFIL)
Authority record attributes (parameter identifier: MQIACF_CHLAUTH_ATTRS). You can specify the following value in the attribute list on its own. This is the default value if the parameter is not specified.

MQIACF_ALL
All attributes.

If MQIACF_ALL is not specified, specify a combination of the following values:

MQCA_ALTERATION_DATE
Alteration Date.

MQCA_ALTERATION_TIME
Alteration Time.

MQCA_CHLAUTH_DESC
Description.

MQCA_REMOTE_Q_MGR_NAME
Remote partner queue manager name.

MQCACH_CHANNEL_NAME
Channel name or pattern.

MQCACH_CLIENT_USER_ID
Client asserted user ID.

MQCACH_CONNECTION_NAME
IP address filter.

MQCACH_CONNECTION_NAME_LIST
A list of IP address patterns.

MQCACH_MCA_USER_ID
MCA User ID mapped on the record.

MQCACH_MCA_USER_ID_LIST
A list of user IDs that are blocked.

MQIACF_CHLAUTH_TYPE
The type of channel authentication record.

MQIACH_MATCH
The type of matching to be applied.

MQIACH_USER_SOURCE
The source of the user ID for this record.

MQIACH_WARNING
Warning mode.

ClntUser (MQCFST)
The client asserted user ID to be matched (parameter identifier: MQCACH_CLIENT_USER_ID). This parameter is valid only when Match is MQMATCH_RUNCHECK.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition.

If you specify an integer filter, you cannot also specify a string filter using the StringFilterCommand parameter.

Match Indicates the type of matching to be applied (parameter identifier MQIACH_MATCH).

You can specify any one of the following values:

MQMATCH_RUNCHECK
A specific match is made against the supplied channel name and optionally supplied Address, QMName, and ClntUser attributes to find the channel authentication record that is matched by the channel at runtime if it connects into this queue manager. If the record discovered has Warn set to MQWARN_YES, a second record might also be displayed to show the actual record the channel uses at runtime. The channel name supplied in this case cannot be generic. This option must be combined with Type MQCAUT_ALL.

MQMATCH_EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MQMATCH_GENERIC.

MQMATCH_GENERIC
A trailing asterisks in the channel profile name is treated as a wild card. If there are no asterisks in the channel profile name, this returns the same output as MQMATCH_EXACT. For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

MQMATCH_ALL
Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more specific matches exist.

QMName (MQCFST)
The name of the remote partner queue manager to be matched (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

This parameter is valid only when Match is MQMATCH_RUNCHECK. The value cannot be generic.

StringFilterCommand (MQCFSF)
String filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. If you specify a string filter, you cannot also specify an integer filter using the IntegerFilterCommand parameter.

Type (MQCFIN)
The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. MQCAUT_USERMAP This channel authentication record maps asserted user IDs to MCAUSER values.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values.

MQCAUT_ALL
Inquire on all types of record. This is the default value.

Inquire Channel Listener

The Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command inquires about the attributes of existing WebSphere MQ listeners.

Required parameters:

ListenerName

Optional parameters:

IntegerFilterCommand, ListenerAttr, StringFilterCommand, TransportType

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

This is the name of the listener whose attributes are required. Generic listener names are supported. A generic name is a character string followed by an asterisk (*); for example, ABC*. It selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names. The listener name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ChannelAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

ListenerAttrs (MQCFIL)
Listener attributes (parameter identifier: MQIACF_LISTENER_ATTRS).

The attribute list might specify this attribute on its own (this is the default value if the parameter is not specified):

MQIACF_ALL
All attributes.

Or it can be a combination of these attributes:
MQCA_ALTERATION_DATE
Date on which the definition was last altered.
MQCA_ALTERATION_TIME
Time at which the definition was last altered.
MQCACH_IP_ADDRESS
IP address for the listener.
MQCACH_LISTENER_DESC
Description of listener definition.
MQCACH_LISTENER_NAME
Name of listener definition.
MQIACH_BACKLOG
number of concurrent connection requests that the listener supports.
MQIACH_LISTENER_CONTROL
Specifies when the queue manager should start and stop the listener.
MQIACH_PORT
Port number.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ChannelAttrs except MQCACH_CHANNEL_NAME and MQCACH_MCA_NAME. Use this to restrict the output from the command by specifying a filter condition.

TransportType (MQCFIN)
Transport protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

If you do not specify this parameter, or if you specify it with a value of MQXPT_ALL, information about all listeners is returned. The value can be:
MQXPT_ALL
All transport types.
MQXPT_TCP
Transmission Control Protocol/Internet Protocol (TCP/IP).

Inquire Channel Listener Status

The Inquire Channel Listener Status (MQCMD_INQUIRE_LISTENER_STATUS) command inquires about the status of one or more WebSphere MQ listener instances. You must specify the name of a listener for which you want to receive status information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either of these:

- Status information for all listener definitions, by using a single asterisk (*).
- Status information for one or more listeners that match the specified name.

Required parameters:
ListenerName

Optional parameters:
IntegerFilterCommand, ListenerStatusAttrs, StringFilterCommand

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME). Generic listener names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names. The listener name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ListenerStatusAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

ListenerStatusAttrs (MQCFIL)
Listener status attributes (parameter identifier: MQIACF_LISTENER_STATUS_ATTRS).

The attribute list can specify this attribute on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

Or it can be a combination of these attributes:
MQCACH_IP_ADDRESS
Listener's IP address.
MQCACH_LISTENER_DESC
Description of listener definition.
MQCACH_LISTENER_NAME
Name of listener definition.
MQCACH_LISTENER_START_DATE
The date on which the listener was started.
MQCACH_LISTENER_START_TIME
The time at which the listener was started.
MQIACH_BACKLOG
Number of concurrent connection requests that the listener supports.
MQIACH_LISTENER_CONTROL
How the listener is to be started and stopped.
MQIACH_LISTENER_STATUS
Current status of the listener.
MQIACH_PORT
Port number for TCP/IP.
MQIACH_XMIT_PROTOCOL_TYPE
Transport type.
MQIACF_PROCESS_ID
The listener CICS task number.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ListenerStatusAttrs except

Inquire Channel Names

The Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command inquires a list of WebSphere MQ channel names that match the generic channel name, and the optional channel type specified.

This PCF is supported on all platforms.

Required parameters:

ChannelName

Optional parameters:

ChannelType

The Inquire Channel Names command, if successful, generates a data response. For details of the Inquire Channel Names response, refer to "Data responses to commands" on page 381.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If present, this parameter limits the channel names returned to channels of the specified type.

The value can be:

MQCHT_SENDER
Sender.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_ALL
All types.

The default value if this parameter is not specified is MQCHT_ALL, which means that channels of all types are eligible.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any

integer type parameter allowed in ChannelAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ChannelAttrs except MQCACH_CHANNEL_NAME and MQCACH_MCA_NAME. Use this to restrict the output from the command by specifying a filter condition.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Channel Status

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command inquires about the status of one or more channel instances.

You must specify the name of the channel for which you want to inquire status information. This can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:
- Status information for all channels, or
- Status information for one or more channels that match the specified name.

You must also specify whether you want:
- The current status data (of current channels only), or
- The saved status data of all channels.

Status for all channels that meet the selection criteria is given, whether the channels were defined manually or automatically.

There are two classes of data available for channel status. These are saved and current. The status fields available for saved data are a subset of the fields available for current data and are called common status fields. Note that although the common data fields are the same, the data values might be different for saved and current status. The rest of the fields available for current data are called current-only status fields.

Saved data consists of the common status fields. This data is reset at the following times:

For all channels:

- When the channel enters or leaves STOPPED state.

For a sending channel:

- Before requesting confirmation that a batch of messages has been received When confirmation has been received
- When confirmation has been received.

For a receiving channel:

- Just before confirming that a batch of messages has been received

For a server connection channel:

- No data is saved.

Therefore, a channel which has never been current will not have any saved status.

Current data consists of the common status fields and current-only status fields. The data fields are continually updated as messages are sent or received.

This method of operation has the following consequences:

- An inactive channel might not have any saved status if it has never been current or has not yet reached a point where saved status is reset.
- The "common" data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can be current or inactive:

Current channels

These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They may not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have current status and can also have saved status.

The term Active is used to describe the set of current channels which are not stopped.

Inactive channels

These are channels that have either not been started or on which a client has not connected, or that have finished or disconnected normally. Inactive channels have either saved status or no status at all.

There can be more than one instance of a receiver server-connection channel current at the same time. For channels of other types, there can only be one instance current at any time.

Required parameters:

ChannelName

Optional parameters:

ChannelInstanceAttrs, ChannelInstanceType, ConnectionName, IntegerFilterCommand, StringFilterCommand, XmitQName

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelInstanceAttrs (MQCFIL)

Channel instance attributes (parameter identifier: MQIACH_CHANNEL_INSTANCE_ATTRS).

If status information is requested which is not relevant for the particular channel type, this is not an error. Similarly, it is not an error to request status information that is applicable only to active channels for saved channel instances. In both of these cases, no structure is returned in the response for the information concerned.

For a saved channel instance, the MQCACH_CURRENT_LUWID, MQIACH_CURRENT_MSGS, and MQIACH_CURRENT_SEQ_NUMBER attributes have meaningful information only if the channel instance is in doubt. However, the attribute values are still returned when requested, even if the channel instance is not in-doubt.

The attribute list might specify the following on its own:

- MQIACF_ALL All attributes.

This is the default value used if the parameter is not specified or it can specify a combination of the following:

Relevant for common status

The following information applies to all sets of channel status, whether or not the set is current.

MQCACH_CHANNEL_NAME
Channel name.

MQCACH_CONNECTION_NAME
Connection name.

MQCACH_CURRENT_LUWID
Logical unit of work identifier for current batch.

MQCACH_LAST_LUWID
Logical unit of work identifier for last committed batch.

MQCACH_XMIT_Q_NAME
Transmission queue name.

MQIACH_CHANNEL_INSTANCE_TYPE
Channel instance type.
MQIACH_CHANNEL_TYPE
Channel type.
MQIACH_CURRENT_MSGS
Number of messages sent or received in current batch.
MQIACH_CURRENT_SEQ_NUMBER
Sequence number of last message sent or received.
MQIACH_INDOUBT_STATUS
Whether the channel is currently in-doubt.
MQIACH_LAST_SEQ_NUMBER
Sequence number of last message in last committed batch.

MQCACH_CURRENT_LUWID, MQCACH_LAST_LUWID, MQIACH_CURRENT_MSGS, MQIACH_CURRENT_SEQ_NUMBER, MQIACH_INDOUBT_STATUS and MQIACH_LAST_SEQ_NUMBER do not apply to server-connection channels, and no values are returned. If specified on the command, they are ignored.

Relevant for current-only status

The following information applies only to current channel instances. The information applies to all channel types, except where stated.
MQCA_Q_MGR_NAME
Name of the queue manager that owns the channel instance.
MQCA_REMOTE_Q_MGR_NAME
Queue manager name of the remote system. The remote queue manager name is always returned regardless of the instance attributes requested.
MQCACH_CHANNEL_START_DATE
Date channel was started.
MQCACH_CHANNEL_START_TIME
Time channel was started.
MQCACH_LAST_MSG_DATE
Date last message was sent, or MQI call was handled.
MQCACH_LAST_MSG_TIME
Time last message was sent, or MQI call was handled.
MQCACH_LOCAL_ADDRESS
Local communications address for the channel.
MQCACH_MCA_USER_ID
The user ID used by the MCA.
MQCACH_SSL_SHORT_PEER_NAME
SSL short peer name.
MQCACH_SSL_CERT_ISSUER_NAME
The full Distinguished Name of the issuer of the remote certificate.
MQIA_MONITORING_CHANNEL
Current level of monitoring data collection.
MQIACF_MONITORING
All channel status monitoring attributes. These are:
- MQIA_MONITORING_CHANNEL
- MQIACH_BATCH_SIZE_INDICATOR
- MQIACH_EXIT_TIME_INDICATOR

- MQIACH_NETWORK_TIME_INDICATOR
- MQIACH_XMITQ_MSGS_AVAILABLE
- MQIACH_XMITQ_TIME_INDICATOR

MQIACH_BUFFERS_RCVD
Number of buffers received.

MQIACH_BUFFERS_SENT
Number of buffers sent.

MQIACH_BYTES_RCVD
Number of bytes received.

MQIACH_BYTES_SENT
Number of bytes sent.

MQIACH_CHANNEL_SUBSTATE
Current channel substate.

MQIACH_CURRENT_SHARING_CONVS
Requests information on the current number of conversations on this channel instance. This attribute applies only to TCP/IP server-connection channels.

MQIACH_EXIT_TIME_INDICATOR
Exit time.

MQIACH_LONG_RETRIES_LEFT
Number of long retry attempts remaining.

MQIACH_MAX_SHARING_CONVS
Requests information on the maximum number of conversations on this channel instance. This attribute applies only to TCP/IP server-connection channels.

MQIACH_MCA_STATUS
MCA status.

MQIACH_MSGS
Number of messages sent or received, or number of MQI calls handled.

MQIACH_NETWORK_TIME_INDICATOR
Network time.

MQIACH_SHORT_RETRIES_LEFT
Number of short retry attempts remaining.

MQIACH_SSL_KEY_RESETS
Number of successful SSL key resets.

MQIACH_SSL_KEY_RESET_DATE
Date of previous successful SSL secret key reset.

MQIACH_SSL_KEY_RESET_TIME
Time of previous successful SSL secret key reset.

MQIACH_STOP_REQUESTED
Whether user stop request has been received.

MQIACH_XMITQ_MSGS_AVAILABLE
Number of messages available to the channel on the transmission queue.

MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

MQIACH_BATCH_SIZE
Batch size.

You cannot use these attribute parameters to filter on:

MQIACF_MONITORING
All channel status monitoring attributes.

Note: You cannot use MQIACF_MONITORING as a parameter to filter on.

MQIACH_BATCH_SIZE_INDICATOR
Batch size.
MQIACH_EXIT_TIME_INDICATOR
Exit time.
MQIACH_MCA_STATUS
MCA status.
MQIACH_NETWORK_TIME_INDICATOR
Network time.
MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

The following attributes do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored:
- MQIACH_BATCH_SIZE_INDICATOR
- MQIACH_BATCH_SIZE
- MQIACH_BATCHES
- MQIACH_LONG_RETRIES_LEFT
- MQIACH_NETWORK_TIME
- MQIACH_NPM_SPEED
- MQCA_REMOTE_Q_MGR_NAME
- MQIACH_SHORT_RETRIES_LEFT
- MQIACH_XMITQ_MSGS_AVAILABLE
- MQIACH_XMITQ_TIME_INDICATOR

The following attributes apply only to server-connection channels. If specified on the command for other types of channel the attribute is ignored and no value is returned:
- MQIACH_CURRENT_SHARING_CONVS
- MQIACH_MAX_SHARING_CONVS

ChannelInstanceType (MQCFIN)
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

It is always returned regardless of the channel instance attributes requested.

You cannot use MQIACH_CHANNEL_INSTANCE_TYPE as a parameter to filter on.

The value can be:

MQOT_CURRENT_CHANNEL
Current channel status.

This is the default, and indicates that only current status information for active channels is to be returned.

Both common status information and active-only status information can be requested for current channels.

MQOT_SAVED_CHANNEL
Saved channel status.

Specify this to cause saved status information for both active and inactive channels to be returned.

Only common status information can be returned. Active-only status information is not returned for active channels if this keyword is specified.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME). If this parameter is present, eligible channel instances are limited to those using this connection name. If it is not specified, eligible channel instances are not limited in this way.

The connection name is always returned, regardless of the instance attributes requested.

The value returned for ConnectionName might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using ConnectionName for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if ConnectionName in the channel definition:

- Is blank or is in "host name" format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number, but the saved channel status value does not.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ChannelInstanceAttrs except MQIACF_ALL and others as noted. Use this to restrict the output from the command by specifying a filter condition.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ChannelInstanceAttrs except MQCACH_CHANNEL_NAME and others as noted. Use this to restrict the output from the command by specifying a filter condition.

If you specify a string filter for ConnectionName or XmitQName, you cannot also specify the ConnectionName or XmitQName parameter.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

If this parameter is present, eligible channel instances are limited to those using this transmission queue. If it is not specified, eligible channel instances are not limited in this way.

The transmission queue name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error codes

In addition to the values for any command shown in section “Error codes applicable to all commands” on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type is invalid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Connection

The Inquire connection (MQCMD_INQUIRE_CONNECTION) command inquires about the applications which are connected to the queue manager, the status of any transactions that those applications are running, and the objects which the application has open.

Required parameters

ConnectionId or GenericConnectionId

Optional parameters

ConnectionAttrs, ConnInfoType, IntegerFilterCommand, StringFilterCommand

Required parameters

ConnectionId (MQCFBS)
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This is the unique connection identifier associated with an application that is connected to the queue manager. Specify either this parameter or GenericConnectionId.

All connections are assigned a unique identifier by the queue manager regardless of how the connection is established.

If you need to specify a generic connection identifier, use the GenericConnectionId parameter instead.

The length of the string is MQ_CONNECTION_ID_LENGTH.

GenericConnectionId (MQCFBS)
Generic specification of a connection identifier (parameter identifier: MQBACF_GENERIC_CONNECTION_ID).

Specify either this parameter or ConnectionId.

If you specify a byte string of zero length, or one which contains only null bytes, information about all connection identifiers is returned. This is the only value permitted for GenericConnectionId.

The length of the string is MQ_CONNECTION_ID_LENGTH.

Optional parameters

ConnectionAttrs (MQCFIL)
Connection attributes (parameter identifier: MQIACF_CONNECTION_ATTRS). The attribute list can specify the following on its own (this is the default value if the parameter is not specified):

MQIACF_ALL
All attributes of the selected ConnInfoType.

or, if you select a value of MQIACF_CONN_INFO_CONN for ConnInfoType, a combination of the following:

MQBACF_CONNECTION_ID
Connection identifier.

MQCACF_APPL_TAG
Name of an application that is connected to the queue manager.

MQCACF_TASK_NUMBER
A 7-digit CICS task number.

MQCACF_TRANSACTION_ID
A 4-character CICS transaction identifier.

MQCACF_UOW_START_DATE
Date on which the transaction associated with the current connection was started.

MQCACF_UOW_START_TIME
Time at which the transaction associated with the current connection was started.

MQCACF_USER_IDENTIFIER
User identifier of the application that is connected to the queue manager.

MQCACH_CHANNEL_NAME
Name of the channel associated with the connected application.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIA_APPL_TYPE
Type of the application that is connected to the queue manager.

MQIACF_CONNECT_OPTIONS
Connect options currently in force for this application connection.

MQIACF_UOW_STATE
State of the unit of work.

MQIACF_UOW_TYPE
Type of external unit of recovery identifier as understood by the queue manager.

or, if you select a value of MQIACF_CONN_INFO_HANDLE for ConnInfoType, a combination of the following:

MQCACF_OBJECT_NAME
Name of each object that the connection has open.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIACF_HANDLE_STATE
Whether an API call is in progress.

MQIACF_OBJECT_TYPE
Type of each object that the connection has open.
MQIACF_OPEN_OPTIONS
Options used by the connection to open each object.

or, if you select a value of MQIACF_CONN_INFO_ALL for ConnInfoType, any of the above.

ConnInfoType (MQCFIN)
Type of connection information to be returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value can be:
MQIACF_CONN_INFO_CONN
Connection information. This is the default value used if the parameter is not specified.
MQIACF_CONN_INFO_HANDLE
Information pertaining only to those objects opened by the specified connection.
MQIACF_CONN_INFO_ALL
Connection information and information about those objects that the connection has open.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ConnectionAttrs except as noted and MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ConnectionAttrs. Use this to restrict the output from the command by specifying a filter condition.

Error codes

This command might return the following in the response format header, in addition to the values applicable to all commands.

Reason (MQLONG)

The value can be:

MQRCCF_CONNECTION_ID_ERROR
Connection identifier not valid.

Inquire Namelist

The Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command inquires about the attributes of namelist objects.

Required parameters

NamelistName

Optional parameters

IntegerFilterCommand, NamelistAttrs, StringFilterCommand

The Inquire Namelist command, if successful, generates a data response. For details of the Inquire Namelist response, refer to “Data responses to commands” on page 381.

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all namelists having names that start with the selected character string.

An asterisk on its own matches all possible names.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in NamelistAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

NamelistAttrs (MQCFIL)
Namelist attributes (parameter identifier: MQIACF_NAMELIST_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of:

MQCA_ALTERATION_DATE
The date on which the information was last altered, in the form yyyy-mm-dd.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form hh.mm.ss.

MQCA_NAMELIST_DESC
The namelist description.

MQCA_NAMES
The namelist names.

MQIA_NAME_COUNT
The number of namelist names.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in NamelistAttrs except MQCA_NAMELIST_NAME. Use this to restrict the output from the command by specifying a filter condition.

Error codes:

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_SELECTOR_ERROR (2067, X'813')
Attribute selector not valid.

MQRC_UNKNOWN_OBJECT_NAME (2085, X'825')
Unknown object name.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Namelist Names

The Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command inquires a list of WebSphere MQ namelists objects that match the generic namelist name

Required parameters

NamelistName

Optional parameters

None.

The Inquire Namelist Names command, if successful, generates a data response. For details of the Inquire Namelist Names response, refer to "Data responses to commands" on page 381.

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string.

An asterisk on its own matches all possible names.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Error codes:

In addition to the values for any command shown in "Error codes applicable to all commands" on page 241for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue

The Inquire Queue (MQCMD_INQUIRE_Q) command inquires about the attributes of WebSphere MQ queues.

This PCF is supported on all platforms.

Required parameters:

QName

Optional parameters:

IntegerFilterCommand, QAttrs, Qtype, StringFilterCommand

The Inquire Queue command, if successful, generates a data response. For details of the Inquire Queue response, refer to "Data responses to commands" on page 381.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in QAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition. If you specify an integer filter for Qtype, you cannot also specify the Qtype parameter.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, the queue identified by QName must be of the specified type.

If this parameter is not present (or if MQQT_ALL is specified), the queue identified by QName can be of any type.

The value can be:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

QAttrs (MQCFIL)
Queue attributes (parameter identifier: MQIACF_Q_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of:

MQCA_ALTERATION_DATE
The date on which the information was last altered, in the form yyyy-mm-dd.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form hh.mm.ss.

MQCA_BASE_Q_NAME
Name of queue that alias resolves to.

MQCA_CICS_FILE_NAME
CSD file name for queue messages.

MQCA_CREATION_DATE
Queue creation date.

MQCA_CREATION_TIME
Queue creation time.

MQIA_ACCOUNTING_Q
Queue accounting setting.

MQIA_DEFINITION_TYPE
Queue definition type.

MQIA_DEF_PERSISTENCE
Default persistence.

MQIA_INHIBIT_GET
Whether get operations are allowed.

MQIA_INHIBIT_PUT
Whether put operations are allowed.

MQIA_MAX_GLOBAL_LOCKS
Buffer size for queue manager to manage concurrent queue access.

MQIA_MAX_LOCAL_LOCKS
Buffer size for applications to manage concurrent queue access.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_MAX_Q_DEPTH
Maximum number of messages allowed on queue.

MQIA_MAX_Q_TRIGGERS
Maximum number of concurrent trigger instances for a particular queue.

MQIA_MONITORING_Q
Queue monitoring setting.

MQIA_PROPERTY_CONTROL
Property control attribute.

MQIA_Q_DEPTH_HIGH_EVENT
Control attribute for queue depth high events.

MQIA_Q_DEPTH_HIGH_LIMIT
High limit for queue depth.

MQIA_Q_DEPTH_LOW_EVENT
Control attribute for queue depth low events.

MQIA_Q_DEPTH_LOW_LIMIT
Low limit for queue depth.

MQIA_Q_DEPTH_MAX_EVENT
Control attribute for queue depth max events.

MQCA_Q_NAME
Queue name.

MQIA_Q_SERVICE_INTERVAL
Limit for queue service interval.

MQIA_Q_SERVICE_INTERVAL_EVENT
Control attribute for queue service interval events.

MQIA_Q_TYPE
Queue type.

MQCA_Q_DESC
Queue description.

MQIA_Q_USERS
Maximum number of active opens to any particular queue.

MQCA_REMOTE_Q_NAME
Name of remote queue as known locally on the remote queue manager.

MQCA_REMOTE_Q_MGR_NAME
Name of remote queue manager.

MQIA_SHAREABILITY
Whether queue can be shared.

MQIA_STATISTICS_Q
Queue statistics setting.

MQCA_TRIGGER_CHANNEL_NAME
Channel name for MCA trigger process.

MQIA_TRIGGER_CONTROL
Trigger control.

MQCA_TRIGGER_DATA
Trigger data.

MQCA_TRIGGER_PROGRAM_NAME
Program name for trigger process.

MQIA_TRIGGER_RESTART
Indicator for the reactivation of a trigger process.

MQCA_TRIGGER_TERM_ID
Terminal identifier for trigger process.

MQCA_TRIGGER_TRANS_ID
Transaction identifier for trigger process.

MQIA_TRIGGER_TYPE
Trigger type.

MQIA_USAGE
Usage.

MQCA_XMIT_Q_NAME
Transmission queue name.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in QAttrs except MQCA_Q_NAME. Use this to restrict the output from the command by specifying a filter condition.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue Manager

The Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command inquires about the attributes of a queue manager.

This PCF is supported on all platforms.

Required parameters:

None

Optional parameters:

QMgrAttrs

The Inquire Queue Manager command, if successful, generates a data response. For details of the Inquire Queue Manager response, refer to "Data responses to commands" on page 381.

Optional parameters

QMgrAttrs (MQCFIL)

Queue manager attributes (parameter identifier: MQIACF_Q_MGR_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of:

MQIA_ACCOUNTING_CONN_OVERRIDE
Accounting connection override setting.

MQIA_ACCOUNTING_INTERVAL
Accounting message interval.

MQIA_ACCOUNTING_MQI
MQI Accounting setting.

MQIA_ACCOUNTING_Q
Default queue accounting setting.

MQCA_ALTERATION_DATE
Date at which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_Q_MGR_IDENTIFIER
Internally generated unique queue manager name.

MQIA_AUTHORITY_EVENT
Control attribute for authority events.

MQIA_BATCH_INTERFACE_AUTO
Indicator for the automatic activation of the batch interface.

MQCA_BATCH_INTERFACE_ID
Batch interface identifier.

MQIA_CHANNEL_AUTO_DEF
Controls whether receiver and server-connection channels can be auto-defined.

MQIA_CHANNEL_AUTO_DEF_EVENT
Controls whether channel auto-definition events are generated.

MQIA_COMMAND_EVENT
Controls attribute for command events.

MQIA_CONFIGURATION_EVENT
Controls attribute for configuration events.

MQIA_MAX_PROPERTIES_LENGTH
Maximum properties length.

MQIA_MONITORING_CHANNEL
Default channel monitoring setting.

MQIA_MONITORING_Q
Default queue monitoring setting.

MQIA_SSL_EVENT
Controls attribute for SSL events.

MQIA_SSL_RESET_COUNT
SSL key reset count.

MQIA_STATISTICS_CHANNEL
Default channel statistics setting.

MQIA_STATISTICS_INTERVAL
Statistics message interval.

MQIA_STATISTICS_MQI
MQI Statistics setting.

MQIA_STATISTICS_Q
Default queue statistics setting.

MQCA_CHANNEL_AUTO_DEF_EXIT
Channel auto-definition exit name.

MQIA_CODED_CHAR_SET_ID
Coded character set identifier.

MQCA_COMMAND_INPUT_Q_NAME
System command input queue name.

MQCA_COMMAND_REPLY_Q_NAME
WebSphere MQ command reply queue.

MQIA_CMD_SERVER_AUTO
Indicator for the automatic activation of the PCF command server.

MQIA_CMD_SERVER_CONVERT_MSG
Indicator for the data conversion of PCF messages.

MQIA_CMD_SERVER_DLQ_MSG
Indicator for the storage of undeliverable PCF reply messages to the system dead letter queue.

MQIA_COMMAND_LEVEL
Command level supported by queue manager.

MQCA_DEAD_LETTER_Q_NAME
Name of dead-letter queue.

MQIA_DIST_LISTS
Distribution list support.

MQIA_INHIBIT_EVENT
Control attribute for inhibit events.

MQIA_LISTENER_PORT_NUMBER
Port number for TCP/IP Listener process.

MQIA_LOCAL_EVENT
Control attribute for local events.

MQIA_MAX_GLOBAL_LOCKS
Buffer size for queue manager to manage concurrent queue access.

MQIA_MAX_HANDLES
Maximum number of handles.

MQIA_MAX_LOCAL_LOCKS
Buffer size for applications to manage concurrent queue access.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_MAX_OPEN_Q
Maximum number of concurrently open queues

MQIA_MAX_Q_DEPTH
Maximum queue depth.

MQIA_PLATFORM
Platform on which the queue manager resides.

MQCA_Q_MGR_DESC
Queue manager description.

MQCA_Q_MGR_NAME
Name of local queue manager.

MQIA_Q_USERS
Maximum number of active opens to any particular queue.

MQIA_REMOTE_EVENT
Control attribute for remote events.

MQIA_START_STOP_EVENT
Control attribute for start stop events.

MQIA_PERFORMANCE_EVENT
Control attribute for performance events.

MQIA_MONITOR_INTERVAL
Queue manager housekeeping process interval.

MQCA_MONITOR_Q_NAME
MQI monitor queue name.

MQCA_SSL_KEY_LIBRARY
SSL key library name.

MQCA_SSL_KEY_MEMBER
SSL key member name.

MQIA_SYNCPOINT
Syncpoint availability.

MQCA_SYSTEM_LOG_Q_NAME
System log queue name.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue Names

The Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command inquires a list of queue names that match the generic queue name, and the optional queue type specified.

This PCF is supported on all platforms.

Required parameters:

QName

Optional parameters:

QType

The Inquire Queue Names command, if successful, generates a data response. For details of the Inquire Queue Names response, refer to "Data responses to commands" on page 381.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If present, this parameter limits the queue names returned to queues of the specified type. If this parameter is not present, queues of all types are eligible. The value may be:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue Status

The Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command inquires about the status of a local WebSphere MQ queue. You must specify the name of a local queue for which you want to receive status information.

Required parameters:

QName

Optional parameters:

IntegerFilterCommand, OpenType, QstatusAttrs, StatusType, StringFilterCommand,

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in QStatusAttrs except MQIACF_ALL, MQIACF_MONITORING, and MQIACF_Q_TIME_INDICATOR. Use this to restrict the output from the command by specifying a filter condition.

OpenType (MQCFIN)
Queue status open type (parameter identifier: MQIACF_OPEN_TYPE). It is always returned, regardless of the queue instance attributes requested.

The value can be:

MQQSOT_ALL
Selects status for queues that are open with any type of access.

MQQSOT_INPUT
Selects status for queues that are open for input.

MQQSOT_OUTPUT
Selects status for queues that are open for output.

The default value if this parameter if not specified is MQQSOT_ALL.

QStatusAttrs (MQCFIL)

Queue status attributes (parameter identifier: MQIACF_Q_STATUS_ATTRS). The attribute list can specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

- Where StatusType is MQIACF_Q_STATUS:

  MQCA_Q_NAME
  Queue name.

  MQCACF_LAST_GET_DATE
  Date of the last message successfully destructively read from the queue.

  MQCACF_LAST_GET_TIME
  Time of the last message successfully destructively read from the queue.

  MQCACF_LAST_PUT_DATE
  Date of the last message successfully put to the queue.

  MQCACF_LAST_PUT_TIME
  Time of the last message successfully put to the queue.

  MQIA_CURRENT_Q_DEPTH
  The current number of messages on the queue.

  MQIA_MONITORING_Q
  Current level of monitoring data collection.

  MQIA_OPEN_INPUT_COUNT
  The number of handles that are currently open for input for the queueThis does not include handles that are open for browse.

  MQIA_OPEN_OUTPUT_COUNT
  The number of handles that are currently open for output for the queue.

  MQIACF_HANDLE_STATE
  Whether an API call is in progress.

  MQIACF_MONITORING
  All of the queue status monitoring attributes. These are:
  – MQCACF_LAST_GET_DATE
  – MQCACF_LAST_GET_TIME
  – MQCACF_LAST_PUT_DATE
  – MQCACF_LAST_PUT_TIME
  – MQIA_MONITORING_Q
  – MQIACF_OLDEST_MSG_AGE
  – MQIACF_Q_TIME_INDICATOR

  MQIACF_OLDEST_MSG_AGE
  Age of oldest message on the queue.

  MQIACF_Q_TIME_INDICATOR
  Indicator of the time that messages remain on the queue.

  MQIACF_UNCOMMITTED_MSGS
  Whether there are uncommitted messages on the queue.

- Where StatusType is MQIACF_Q_HANDLE:

  MQCA_Q_NAME
  Queue name.

  MQCACF_APPL_TAG
  This is a string containing the tag of the application connected to the queue manager.

  MQCACF_TASK_NUMBER
  CICS task number.

  MQCACF_TRANSACTION_ID
  CICS transaction identifier.

  MQCACF_USER_IDENTIFIER
  The username of the application that has opened the specified queue.

  MQCACH_CHANNEL_NAME
  The name of the channel that has the queue open, if any.

  MQCACH_CONNECTION_NAME
  The connection name of the channel that has the queue open, if any.

  MQIA_APPL_TYPE
  The type of application that has the queue open.

  MQIACF_OPEN_BROWSE
  Open browse.

  MQIACF_OPEN_INPUT_TYPE
  Open input type.

  MQIACF_OPEN_INQUIRE
  Open inquire.

  MQIACF_OPEN_OPTIONS
  The options used to open the queue.

  If this parameter is requested, the following parameter structures are also returned:
  - OpenBrowse
  - OpenInputType
  - OpenInquire
  - OpenOutput
  - OpenSet

  MQIACF_OPEN_OUTPUT
  Open output.

  MQIACF_OPEN_SET
  Open set.

  MQIACF_UOW_TYPE
  Type of external unit of recovery identifier as seen by the queue manager.

StatusType (MQCFIN)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information required. The value can be:

MQIACF_Q_STATUS
Selects status information relating to queues.

MQIACF_Q_HANDLE
Selects status information relating to the handles that are accessing the queues.

The default value, if this parameter is not specified, is MQIACF_Q_STATUS.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in QStatusAttrs except MQCA_Q_NAME. Use this to restrict the output from the command by specifying a filter condition.

Error codes

In addition to the values for any command shown in "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

- Reason (MQLONG)

The value can be:
MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.
MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.
MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.
MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.
MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.
MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.
MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.
MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.
MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.
MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.
MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.
MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.
MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.
MQRCCF_Q_STATUS_NOT_FOUND
No status available for specified queue.
MQRCCF_Q_TYPE_ERROR
Queue type not valid.
MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.
MQRC_UNKNOWN_OBJECT_NAME
Unknown object name.
MQRC_UNKNOWN_Q_NAME
Unknown queue name.

Inquire Service

The Inquire Service (MQCMD_INQUIRE_SERVICE) command inquires about the attributes of existing WebSphere MQ services.

Required parameters:

ServiceName

Optional parameters:

IntegerFilterCommand, ServiceAttrs, StringFilterCommand

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This is the name of the service whose attributes are required. Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ServiceAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

ServiceAttrs (MQCFIL)
Service attributes (parameter identifier: MQIACF_SERVICE_ATTRS).

The attribute list might specify this attribute on its own (this is the default value if the parameter is not specified):

MQIACF_ALL
All attributes.

Or it can specify one or more of these attributes:
MQCA_ALTERATION_DATE
Date on which the definition was last altered.
MQCA_ALTERATION_TIME
Time at which the definition was last altered.
MQCA_SERVICE_DESC
Description of service definition.
MQCA_SERVICE_NAME
Name of service definition.

MQCA_SERVICE_START_ARGS
Arguments to be passed to the service program.
MQCA_SERVICE_START_COMMAND
CICS transaction to start the service.
MQCA_SERVICE_STOP_ARGS
Arguments to be passed to the service program.
MQIA_SERVICE_CONTROL
When the queue manager should start the service.
MQIA_SERVICE_TYPE
Mode in which the service is to run.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ServiceAttrs except MQCA_SERVICE_NAME. Use this to restrict the output from the command by specifying a filter condition.

Inquire Service Status

The Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command inquires about the status of one or more WebSphere MQ service instances.

Required parameters:
ServiceName

Optional parameters:
IntegerFilterCommand, ServiceStatusAttrs, StringFilterCommand

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCACH_SERVICE_NAME).

Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in ServiceStatusAttrs except MQIACF_ALL. Use this to restrict the output from the command by specifying a filter condition.

ServiceStatusAttrs (MQCFIL)
Service status attributes (parameter identifier: MQIACF_SERVICE_STATUS_ATTRS).

The attribute list can specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

Or it can specify one or more of these attributes:

MQCA_SERVICE_DESC
Description of service definition.
MQCA_SERVICE_NAME
Name of service definition.
MQCA_SERVICE_START_ARGS
The arguments to pass to the service program.
MQCA_SERVICE_START_COMMAND
CICS transaction code to start the service.
MQCA_SERVICE_STOP_ARGS
The arguments to pass to the stop command to stop the service.
MQCA_SERVICE_STOP_COMMAND
The name of the program to run to stop the service.
MQCACF_SERVICE_START_DATE
The date on which the service was started.
MQCACF_SERVICE_START_TIME
The time at which the service was started.
MQIA_SERVICE_CONTROL
How the service is to be started and stopped.
MQIA_SERVICE_TYPE
The mode in which the service is to run.
MQIACF_PROCESS_ID
The CICS task number under which this service is executing. WebSphere MQ for z/VSE always returns 0 for service process ID.
MQIACF_SERVICE_STATUS
Current status of the service provided service type is SERVER.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in ServiceStatusAttrs except MQCA_SERVICE_NAME. Use this to restrict the output from the command by specifying a filter condition.

Ping Queue Manager

The Ping Queue Manager (MQCMD_PING_Q_MGR) command tests whether the queue manager and its command server is responsive to commands. If the queue manager is responding a positive reply is returned.

This PCF is supported on all platforms.

Required parameters:
None

Optional parameters:
None

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

Reset Channel

The Reset Channel (MQCMD_RESET_CHANNEL) command resets the message sequence number for an WebSphere MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

This command can be issued to a channel of type MQCHT_SENDER or MQCHT_RECEIVER. However, if it is issued to a sender (MQCHT_SENDER) channel, the value at both ends (issuing end and receiver end), is reset when the channel is next initiated or resynchronized. The value at both ends is reset to be equal.

If the command is issued to a receiver (MQCHT_RECEIVER) channel, the value at the Sender end is not reset as well; this must be done separately if necessary.

This PCF is supported on all platforms.

Required parameters:
ChannelName

Optional parameters:
MsgSeqNumber

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be reset. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

MsgSeqNumber (MQCFIN)
Message sequence number (parameter identifier: MQIACH_MSG_SEQUENCE_NUMBER).

Specifies the new message sequence number.

The value may be in the range 1-999 999 999. The default value is one.

Error codes

In addition to the values for any command shown in section “Error codes applicable to all commands” on page 241', for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Set Channel Authentication Record

The Set Channel Authentication Record (MQCMD_SET_CHLAUTH_REC) command sets the allowed partner details and mappings to MCAUSER for a channel or set of channels.

Required parameters:

ProfileName

Optional parameters:

Action, Address, AddrList, ClntUser, Description, MCAUser, QMName, Type, UserList, UserSrc, Warn

Required parameters

ProfileName (MQCFST)
The name of the channel or set of channels for which you are setting channel authentication configuration (parameter identifier: MQCACH_CHANNEL_NAME). You can use a trailing asterisk (*) as a wildcard to specify a set of channels. If you set Type to MQCAUT_BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

This table shows which parameters are valid for each value of Action:

Table 9. Action

Parameter	MQACT_ADD	MQACT_REMOVE	MQACT_REMOVEALL
ProfileName	Y	Y	Y
Type	Y	Y	Y
Action	Y	Y	Y
Address	Y	Y	
Addrlist	Y	Y	
ClntUser	Y	Y	
MCAUser	Y		
QMName	Y	Y	
UserList	Y	Y	
UserSrc	Y		
Warn	Y		
Description	Y		

Action (MQCFIN)
The action to perform on the channel authentication record (parameter identifier: MQIACF_ACTION). The following values are valid:

MQACT_ADD
Add the specified configuration to a channel authentication record. This is the default value. For types MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, the command fails. For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration is added to the list.

MQACT_REPLACE
Replace the current configuration of a channel authentication record. For types MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added. For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like MQACT_REMOVEALL.

MQACT_REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like MQACT_REMOVEALL.

MQACT_REMOVEALL
Remove all members of the list and thus the whole record (for MQCAUT_BLOCKADDR and MQCAUT_BLOCKUSER) or all previously defined mappings (for MQCAUT_ADDRESSMAP, MQCAUT_QMGRMAP and MQCAUT_USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in AddrList, UserList, Address,

QMName or ClntUser. If the specified type has no current configuration the command still succeeds.

Address (MQCFST)
The filter to be used to compare with the IP address of the partner queue manager or client at the other end of the channel (parameter identifier: MQCACH_CONNECTION_NAME). This parameter is mandatory when Type is MQCAUT_ADDESSMAP and is also valid when Type is MQCAUT_USERMAP, or MQCAUT_QMGRMAP and Action is MQACT_ADD, MQACT_REPLACE, or MQACT_REMOVE. You can define more than one channel authentication object with the same main identity, for example the same queue manager name, with different addresses. The maximum length of the string is MQ_CONN_NAME_LENGTH.

AddrList (MQCFSL)
A list of up to 56 generic IP addresses which are banned from accessing this queue manager on any channel (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

This parameter is only valid when Type is MQCAUT_BLOCKADDR. The maximum length of each address is MQ_CONN_NAME_LENGTH.

ClntUser (MQCFST)
The client asserted user ID to be mapped to a new user ID or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

This parameter is valid only when Type is MQCAUT_BLOCKADDR.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

Description (MQCFST)
Provides descriptive information about the channel authentication record, which is displayed when you issue the Inquire Channel Authentication Records command (parameter identifier: MQCA_CHLAUTH_DESC).

This parameter must contain only displayable characters. In a DBCS installation, it can contain DBCS characters. The maximum length of the string is MQ_CHLAUTH_DESC_LENGTH.

Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUser (MQCFST)
The user identifier to be used when the inbound connection matches the IP address, client asserted user ID or remote queue manager name supplied (parameter identifier: MQCACH_MCA_USER_ID). This parameter is mandatory when UserSrc is MQUSRC_MAP and is valid when Type is MQCAUT_ADDRESSMAP, MQCAUT_USERMAP, or MQCAUT_QMGRMAP. This parameter is valid only when Action is MQACT_ADD or MQACT_REPLACE. The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

QMName (MQCFST)
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked (parameter identifier: MQCA_REMOTE_Q_MGR_NAME). This parameter is valid only when Type is MQCAUT_QMGRMAP The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Type (MQCFIN)
The type of channel authentication record for which to set allowed partner

details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting. The MQCAUT_BLOCKUSER parameter must be accompanied by a UserList.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses. The MQCAUT_BLOCKADDR parameter must be accompanied by an AddrList.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. The MQCAUT_ADDRESSMAP parameter must be accompanied by an Address.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The MQCAUT_USERMAP parameter must be accompanied by a ClntUser.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values. The MQCAUT_QMGRMAP parameter must be accompanied by a QMName.

UserList (MQCFSL)
A list of up to 100 user IDs which are banned from using this channel or set of channels (parameter identifier: MQCACH_MCA_USER_ID_LIST).

This parameter is only valid when TYPE is MQCAUT_BLOCKUSER. The maximum length of each user ID is MQ_MCA_USER_ID_LENGTH .

UserSrc (MQCFIN)
The source of the user ID to be used for MCAUSER at run time (parameter identifier: MQIACH_USER_SOURCE). The following values are valid:

MQUSRC_MAP
Inbound connections that match this mapping use the user ID specified in the MCAUser attribute.

MQUSRC_NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

MQUSRC_CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field. Note that Warn and MQUSRC_CHANNEL, or MQUSRC_MAP are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

Warn (MQCFIN)
Indicates whether this record operates in warning mode (parameter identifier: MQIACH_WARNING).

MQWARN_NO
This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

MQWARN_YES
This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if events are configured, an event message is created showing the details of what would have been blocked. The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Error codes

This command might return the following error codes in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CHLAUTH_TYPE_ERROR
Channel authentication record type not valid.

MQRCCF_CHLAUTH_ACTION_ERROR
Channel authentication record action not valid.

MQRCCF_CHLAUTH_USERSRC_ERROR
Channel authentication record user source not valid.

MQRCCF_WRONG_CHLAUTH_TYPE
Parameter not allowed for this channel authentication record type.

MQRCCF_CHLAUTH_ALREADY_EXISTS
Channel authentication record already exists

Start Channel

The Start Channel (MQCMD_START_CHANNEL) command starts an WebSphere MQ channel.

This command can be issued to a channel of type MQCHT_SENDER or MQCHT_RECEIVER. Under WebSphere MQ for z/VSE, starting a channel this way makes it available for use.

This PCF is supported on all platforms.

Required parameters:

ChannelName

Optional parameters:

None

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Start Channel Listener

The Start Channel Listener (MQCMD_START_CHANNEL_LISTENER) command starts a WebSphere MQ listener provided the current status is STOPPED.

Required parameters:

ListenerName

Optional parameters:

None

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

The name of the listener definition to be started.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Start Service

The Start Service (MQCMD_START_SERVICE) command starts an existing WebSphere MQ service definition.

Required parameters:

ServiceName

Optional parameters:

None

Required parameters

ServiceName (MQCFST)

Service name (parameter identifier: MQCA_SERVICE_NAME).

This is the name of the service definition to be started.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Stop Channel

The Stop Channel (MQCMD_STOP_CHANNEL) command stops an WebSphere MQ channel.

This command can be issued to a channel of any type supported by WebSphere MQ for z/VSE.

This PCF is supported on all platforms.

Required parameters:

ChannelName

Optional parameters:

ChannelStatus, Quiesce

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelStatus (MQCFIN)

The new state of the channel after the command is executed (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_INACTIVE

Channel is inactive.

MQCHS_STOPPED

Channel is stopped. This is the default if nothing is specified.

Quiesce (MQCFIN)

Quiesce channel (parameter identifier: MQIACF_QUIESCE).

Specifies whether the channel should be quiesced or stopped immediately. If this parameter is not present the channel is quiesced. The value can be:

MQQO_YES
Quiesce the channel.

MQQO_NO
Do not quiesce the channel.

Under WebSphere MQ for z/VSE, the Quiesce parameter is ignored.

Error codes

In addition to the values for any command shown in section "Error codes applicable to all commands" on page 241, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value can be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_QUIESCE_VALUE_ERROR
Quiesce value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Stop Channel Listener

The Stop Channel Listener (MQCMD_STOP_CHANNEL_LISTENER) command stops a WebSphere MQ listener provided the current status is RUNNING.

Required parameters:

ListenerName

Optional parameters:

None

Required parameters

ListenerName (MQCFST)

Listener name (parameter identifier: MQCACH_LISTENER_NAME).

The name of the listener definition to be stopped.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Stop Connection

The Stop Connection (MQCMD_STOP_CONNECTION) command attempts to break a connection between an application and the queue manager. There may be circumstances in which the queue manager cannot implement this command.

Required parameters

ConnectionId

Optional parameters

None.

Required parameters

ConnectionId (MQCFBS)

Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This is the unique connection identifier associated with an application that is connected to the queue manager.

The length of the byte string is MQ_CONNECTION_ID_LENGTH.

Error codes

This command might return the following in the response format header, in addition to the values applicable to all commands.

Reason (MQLONG)

The value can be:

MQRCCF_CONNECTION_ID_ERROR

Connection identifier not valid.

Stop Service

The Stop Service (MQCMD_STOP_SERVICE) command stops an existing WebSphere MQ service definition of type SERVER that is running or service definition of type COMMAND.

Required parameters:

ServiceName

Optional parameters:

None

Required parameters

ServiceName (MQCFST)

Service name (parameter identifier: MQCA_SERVICE_NAME).

This is the name of the service definition to be stopped.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Change Subscription

The Change Subscription (MQCMD_CHANGE_SUBSCRIPTION) command changes the specified attributes of an existing WebSphere MQ subscription. For any optional parameters that are omitted, the value does not change.

Required parameters:

SubName or SubId

Optional parameters:

TopicString, TopicObject

Required parameters

One of:

SubName (MQCFST)

The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME). The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)

The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID). The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

Destination (MQCFST)

Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass (MQCFIN)

Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Whether the destination is managed.

Specify either:

MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the Destination field.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

Provides a correlation identifier that is placed in the CorrelId field of the message descriptor for all the messages sent to this subscription. The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR). If specified it is the name of the local queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

The default value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the AccountingToken field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN). The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationidentifier (MQCFST)
Value of the application identity data used in the ApplIdentityData field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA. The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of the message sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY). The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
Priority of messages sent to this subscription is taken from that supplied to the published message.

This is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
Priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

Value ignored in z/VSE (0 always returned on inquiry).

PublishSubscribeProperties (MQCFIN)
Specifies how publish/suscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_MSGPROP
Publish subscribe properties are added as message properties.

MQPSPROP_NONE
Do not add publish/suscribe properties to the messages. This is the supplied default value.

Requestonly (MQCFIN)
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY). The value can be:
MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.
MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API call.

SubscriptionScope (MQCFIN)
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE). The value can be:
MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.
MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

In WebSphere MQ for z/VSE, although MQTSCOPE_ALL is accepted it is treated as MQTSCOPE_QMGR. Only local queue manager is supported. On inquiry MQTSCOPE_QMGR is always returned.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last tookover the subscription. (parameter identifier: MQCACF_SUB_USER_ID). The maximum length of the string is MQ_USER_ID_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCACF_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH.

VariableUser (MQCFIN)
Specifies whether a user other than the one who created the subscription, that is, the user shown in SubscriptionUser can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:
MQVU_ANY_USER
Any user can take over the ownership. This is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

Copy Subscription

The Copy Subscription (MQCMD_COPY_SUBSCRIPTION) command creates a new WebSphere MQ subscription, using, for attributes not specified in the command, the attribute values of an existing subscription.

Required parameters:

FromSubscriptionName, ToSubscriptionName, SubName or SubId

Optional parameters:

CommandScope, Destination, DestinationClass, DestinationCorrelId, DestinationQueueManager, Expiry, PublishedAccountingToken, PublishedApplicationIdentifier, VariableUser

Required parameters

FromSubscriptionName (MQCFST)
The name of the subscription definition to be copied from (parameter identifier: MQCACF_FROM_SUB_NAME).

The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

ToSubscriptionName (MQCFST)
The name of the subscription to copy to (parameter identifier: MQCACF_TO_SUBSCRIPTION_NAME). The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

You require at least one of SubName or SubId.

SubName (MQCFST)
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME). The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID). The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

Destination (MQCFST)
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass (MQCFIN)
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Whether the destination is managed.

Specify either:
MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the Destination field.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

Provides a correlation identifier that is placed in the CorrelId field of the message descriptor for all the messages sent to this subscription. The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR). If specified it is the name of the local queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

The default value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the AccountingToken field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN). The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationidentifier (MQCFST)
Value of the application identity data used in the ApplIdentityData field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA. The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of the message sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY). The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
Priority of messages sent to this subscription is taken from that supplied to the published message.

This is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
Priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

Value ignored in z/VSE (0 always returned on inquiry).

PublishSubscribeProperties (MQCFIN)
Specifies how publish/suscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_MSGPROP
Publish subscribe properties are added as message properties.

MQPSPROP_NONE
Do not add publish/suscribe properties to the messages. This is the supplied default value.

Requestonly (MQCFIN)
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY). The value can be:
MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.
MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API call.

SubscriptionScope (MQCFIN)
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE). The value can be:
MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.
MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

In WebSphere MQ for z/VSE, although MQTSCOPE_ALL is accepted it is treated as MQTSCOPE_QMGR. Only local queue manager is supported. On inquiry MQTSCOPE_QMGR is always returned.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last tookover the subscription (parameter identifier: MQCACF_SUB_USER_ID). The maximum length of the string is MQ_USER_ID_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCACF_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH.

VariableUser (MQCFIN)
Specifies whether a user other than the one who created the subscription, that is, the user shown in SubscriptionUser can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:
MQVU_ANY_USER
Any user can take over the ownership. This is the supplied default value.
MQVU_FIXED_USER
No other user can take over the ownership.

Create Subscription

The Create Subscription (MQCMD_CREATE_SUBSCRIPTION) command creates a new WebSphere MQ administrative subscription so that existing applications can participate in publish/suscribe application.

Required parameters:

SubName or SubId, TopicString or TopicObject

Optional parameters:

CommandScope, Destination, DestinationClass, DestinationCorrelId, DestinationQueueManager, Expiry, PublishedAccountingToken, PublishedApplicationIdentifier, VariableUser

Required parameters

You must provide the SubName.

SubName (MQCFST)
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME). The maximum length of the string is MQ_SUB_NAME_LENGTH.

You require at least one of TopicObject or TopicString.

TopicObject (MQCFST)
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCACF_TOPIC). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH (in z/VSE is 256.)

Optional parameters

Destination (MQCFST)
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass (MQCFIN)
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Whether the destination is managed.

Specify either:
MQDC_MANAGED
The destination is managed.
MQDC_PROVIDED
The destination queue is as specified in the Destination field.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

Provides a correlation identifier that is placed in the CorrelId field of the message descriptor for all the messages sent to this subscription. The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR). If specified it is the name of the local queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

The default value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the AccountingToken field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN). The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationidentifier (MQCFST)
Value of the application identity data used in the ApplIdentityData field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA. The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of the message sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY). The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
Priority of messages sent to this subscription is taken from that supplied to the published message.

This is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
Priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

Value ignored in z/VSE (0 always returned on inquiry).

PublishSubscribeProperties (MQCFIN)
Specifies how publish/suscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_MSGPROP
Publish subscribe properties are added as message properties.

MQPSPROP_NONE
Do not add publish/suscribe properties to the messages. This is the supplied default value.

Requestonly (MQCFIN)
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY). The value can be:
MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.
MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API call.

SubscriptionScope (MQCFIN)
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE). The value can be:
MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.
MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

In WebSphere MQ for z/VSE, although MQTSCOPE_ALL is accepted it is treated as MQTSCOPE_QMGR. Only local queue manager is supported. On inquiry MQTSCOPE_QMGR is always returned.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last tookover the subscription (parameter identifier: MQCACF_SUB_USER_ID). The maximum length of the string is MQ_USER_ID_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCACF_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH.

VariableUser (MQCFIN)
Specifies whether a user other than the one who created the subscription, that is, the user shown in SubscriptionUser can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:
MQVU_ANY_USER
Any user can take over the ownership. This is the supplied default value.
MQVU_FIXED_USER
No other user can take over the ownership.

Delete Subscription

The Delete Subscription (MQCMD_DELETE_SUBSCRIPTION) command deletes a subscription.

Required parameters:

SubName or SubId

Optional parameters:

None

Required parameters

One of:

SubName (MQCFST)
Subscription name (parameter identifier: MQCACF_SUB_NAME). Specifies the unique subscription name. The subscription name, if provided, must be fully specified; a wildcard is not acceptable. The subscription name must refer to a durable subscription.

If SubName is not provided, SubId must be specified to identify the subscription to be deleted. The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier.

You must supply a value for SubId if you have not supplied a value for SubName.

Inquire Subscription

The Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command inquires about the attributes of a subscription.

Required parameters:

SubId or SubName

Optional parameters:

Durable, SubAttrs, SubType

Required parameters

One of:

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID). Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the SubId is used as the DestinationCorrelId. The maximum length of the string is MQ_CORREL_ID_LENGTH.

SubName (MQCFST)
The application's unique identifier for a subscription (parameter identifier: MQCACF_SUB_NAME). The maximum length of the string is MQ_SUB_NAME_LENGTH.

Optional parameters

Durable (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).
MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed.

MQSUB_DURABLE_NO
Information about nondurable subscriptions only is displayed.
MQSUB_DURABLE_ALL
Information about all subscriptions is displayed.

SubscriptionAttrs (MQCFIL)
Subscription attributes (parameter identifier: MQIACF_SUB_ATTRS).

To select the attributes you want to display you can specify;
- ALL to display all attributes.
- SUMMARY to display a subset of the attributes (see MQIACF_SUMMARY for a list).
- Any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.
MQIACF_SUMMARY
Use this parameter to display:

MQBACF_DESTINATION_CORREL_ID
MQBACF_SUB_ID
MQCACF_DESTINATION
MQCACF_DESTINATION_Q_QMGR
MQCACF_SUB_NAME
MQCACF_TOPIC_STRING
MQIACF_SUB_TYPE

MQBACF_ACCOUNTING_TOKEN
The accounting token passed by the subscriber for propagation into messages sent to this subscription in the AccountingToken field of the MQMD.
MQBACF_CONNECTION_ID
The currently active ConnectionId (CONNID) that has opened this subscription. Used to detect local publications.
MQBACF_DESTINATION_CORREL_ID
The CorrelId used for messages sent to this subscription.
MQBACF_SUB_ID
The internal unique key identifying a subscription.
MQCA_ALTERATION_DATE
The date of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.
MQCA_ALTERATION_TIME
The time of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.
MQCA_CREATION_DATE
The date of the first MQSUB command that caused this subscription to be created.
MQCA_CREATION_TIME
The time of the first MQSUB that caused this subscription to be created.
MQCA_RESUME_DATE
The date of the most recent MQSUB which connected to this subscription.
MQCA_RESUME_TIME
The time of most recent MQSUB which connected to this subscription.
MQCA_TOPICSTRING
The resolved topic string the subscription is for.

MQCACF_APPL_IDENTITY_DATA
The identity data passed by the subscriber for propagation into messages sent to this subscription in the ApplIdentity field of the MQMD.

MQCACF_DESTINATION
The destination for messages published to this subscription.

MQCACF_DESTINATION_Q_MGR
The destination queue manager for messages published to this subscription.

MQCACF_LAST_MSG_TIME
The time at which a message was last sent to the destination specified by this subscription.

MQCACF_LAST_MSG_DATE
The date on which a message was last sent to the destination specified by this subscription.

MQCACF_SUB_NAME
The application's unique identifier for a subscription.

MQCACF_SUB_USER_ID
The userid that owns the subscription. This is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last tookover the subscription.

MQCACF_TOPIC
The name of the topic object that identifies a position in the topic hierarchy to which the topic string is concatenated.

MQIACF_DESTINATION_CLASS
Indicated whether this is a managed subscription.

MQIACF_DURABLE_SUBSCRIPTION
Whether the subscription is durable, persisting over queue manager restart.

z/VSE only has persistent messages.

MQIACF_EXPIRY
The time to live from creation date and time.

MQIACF_MESSAGE_COUNT
The number of messages put to the destination specified by this subscription.

MQIACF_PUB_PRIORITY
The priority of the messages sent to this subscription.

z/VSE always returns 0.

MQIACF_PUBSUB_PROPERTIES
The manner in which publish/subscribe related message properties are added to messages sent to this subscription.

MQIACF_REQUEST_ONLY
Indicates whether the subscriber polls for updates via MQSUBRQ API, or whether all publications are delivered to this subscription.

MQIACF_SUB_TYPE
The type of subscription - how it was created.

MQIACF_SUBSCRIPTION_SCOPE
Whether the subscription forwards messages to all other queue managers directly connected via a pub/sub collective or hierarchy, or the subscription forwards messages on this topic within this queue manager only.

z/VSE always returns MQTSCOPE_QMGR.

MQIACF_VARIABLE_USER_ID
Users other than the creator of this subscription that can connect to it (subject to topic and destination authority checks).

SubscriptionType (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.

MQSUBTYPE_ALL
All subscription types are displayed.

MQSUBTYPE_API
Subscriptions created by applications via an MQ API are displayed.

MQSUBTYPE_USER
USER subscriptions (those with SUBTYPE of either ADMIN or API) are displayed. This is the default value.

Inquire Subscription Status

The Inquire Subscription Status (MQCMD_INQUIRE_SBSTATUS) command inquires about the status of a subscription.

Required parameters:

SubName or SubId

Optional parameters:

ActiveConnection, Durable, LastPublishDate, LastPublishTime, NumberMsgs, ResumeDate, ResumeTime, SubID, SubType, SubscriptionUser

Required parameters

One of:
SubName (MQCFST)
The unique identifier of an application for a subscription (parameter identifier: MQCACF_SUB_NAME). If SubName is not provided, SubId must be specified to identify the subscription to be inquired. The maximum length of the string is MQ_SUB_NAME_LENGTH.
SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID). Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the SubId is used as the DestinationCorrelId. You must supply a value for SubId if you have not supplied a value for SubName. The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

Durable (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed. This is the default.
MQSUB_DURABLE_NO
Information about non-durable subscriptions only is displayed.

SubId (MQCFBS)
Use this attribute to specify the subscription identifier (parameter identifier: MQBACF_SUB_ID) of the subscription you want to display.

SubscriptionType (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).
MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.
MQSUBTYPE_ALL
All subscription types are displayed.
MQSUBTYPE_API
Subscriptions created by applications through a WebSphere MQ API call are displayed.
MQSUBTYPE_USER
USER subscriptions (those with SUBTYPE of either ADMIN or API) are displayed. This is the default value.

StatusAttrs (MQCFIL)
Subscription status attributes (parameter identifier: MQIACF_SDSTATUS_ATTRS).

To select the attributes you want to display you can specify:
- ALL to display all attributes.
- Any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.
MQBACF_CONNECTION_ID
The currently active ConnectionID that has opened the subscription.
MQCACF_DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).
MQSUB_DURABLE_NO
The subscription is removed when the application that created it is closed or disconnected from the queue manager.
MQSUB_DURABLE_YES
The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.
MQCACF_LAST_PUB_DATE
The date that a message was last sent to the destination specified by the subscription.
MQCACF_LAST_PUB_TIME
The time when a message was last sent to the destination specified by the subscription.

MQIACF_MESSAGE_COUNT
The number of messages put to the destination specified by the subscription.
MQCACF_RESUME_DATE
The date of the most recent MQSUB command that connected to the subscription.
MQCACF_RESUME_TIME
The time of the most recent MQSUB command that connected to the subscription.
MQIACF_SUB_TYPE
The type of subscription - how it was created.
MQSUBTYPE_ADMIN
Created using the DEF SUB MQSC or Create SubscriptionPCF command. This Subtype also indicates that a subscription has been modified using an administrative command.
MQSUBTYPE_API
Created using an MQSUB API call.
MQCACF_SUB_USER_ID
The userid owns the subscription.

Change Topic

The Change Topic (MQCMD_CHANGE_TOPIC) command changes the specified attributes of an existing WebSphere MQ administrative topic definition. For any optional parameters that are omitted, the value does not change.

Required parameter:

TopicName

Optional parameters:

DefPersistence, DefPriority, DefPutResponse, DurableModelQName, DurableSubscriptions, InhibitPublications, InhibitSubscriptions, NonDurableModelQName, PersistentMsgDelivery, TopicDesc, TopicType

Required parameter

TopicName (MQCFST)
The name of the administrative topic definition to be changed (parameter identifier: MQCA_TOPIC_NAME). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE). Specifies the default for message-persistence of messages published to the topic. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be:
MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

There are only persistent messages in z/VSE.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY). Specifies the default priority of messages published to the topic. Values are ignored in z/VSE.

Specify either:
integer
The default priority to be used, in the range zero through to the maximum priority value that is supported (9).
MQPRI_PRIORITY_AS_PARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DefPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE). The value can be:
MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.
MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB). The value can be:
MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.
MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.
MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB). The value can be:
MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_PUB_INHIBITED
Publications are inhibited for this topic.
MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB). The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.
MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY). The value can be:
MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.
MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.
MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.
MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC). Text that briefly describes the object

The maximum length is MQ_TOPIC_DESC_LENGTH.

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

TopicType (MQCFIN)
Topic type (parameter identifier: MQIA_TOPIC_TYPE). The value specified must match the type of the topic being changed. The value can be:

MQTOPT_LOCAL
Local topic object

Copy Topic

The Copy Topic (MQCMD_COPY_TOPIC) command creates a WebSphere MQ administrative topic definition by using, for attributes not specified in the command, the attribute values of an existing topic definition.

Required parameters:

FromTopicName, TopicString, ToTopicName

Optional parameters:

DefPersistence, DefPriority, DefPutResponse, DurableModelQName, DurableSubscriptions, InhibitPublications, InhibitSubscriptions, NonDurableModelQName, PersistentMsgDelivery, TopicDesc, TopicType

Required parameters

FromTopicName (MQCFST)
The name of the administrative topic object definition to be copied from (parameter identifier: MQCACF_FROM_TOPIC_NAME). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING). This string uses the forward slash (/) character as a delimiter for elements within the topic tree. The maximum length of the string is MQ_TOPIC_STR_LENGTH (256 in z/VSE).

ToTopicName (MQCFST)
The name of the administrative topic definition to copy to (parameter identifier: MQCACF_TO_TOPIC_NAME). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE). Specifies the default for message-persistence of messages published to the topic. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be:
MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.
MQPER_PERSISTENT
Message is persistent.

There are only persistent messages in z/VSE.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY). Specifies the default priority of messages published to the topic. Values are ignored in z/VSE.

Specify either:
integer
The default priority to be used, in the range zero through to the maximum priority value that is supported (9).
MQPRI_PRIORITY_AS_PARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DefPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE). The value can be:
MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.
MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB). The value can be:
MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.
MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.
MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB). The value can be:
MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_PUB_INHIBITED
Publications are inhibited for this topic.
MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB). The value can be:
MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.
MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY). The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC). Text that briefly describes the object

The maximum length is MQ_TOPIC_DESC_LENGTH.

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

TopicType (MQCFIN)
Topic type (parameter identifier: MQIA_TOPIC_TYPE). The value specified must match the type of the topic being changed. The value can be:

MQTOPT_LOCAL
Local topic object

Create Topic

The Create Topic (MQCMD_CREATE_TOPIC) command creates a WebSphere MQ administrative topic definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters:

TopicName, TopicString

Optional parameters:

DefPersistence, DefPriority, DefPutResponse, DurableModelQName, DurableSubscriptions, InhibitPublications, InhibitSubscriptions, NonDurableModelQName, PersistentMsgDelivery, TopicDesc, TopicType

Required parameters

TopicName (MQCFST)
The name of the administrative topic definition to be created (parameter identifier: MQCA_TOPIC_NAME). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING). This parameter is required and cannot contain the empty string. The "/" character within this string has a special meaning. It delimits the elements in the topic tree. A topic string can start with the "/" character but is not required to. A string starting with the "/" character is not the same as a string that does not start with the "/" character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH (256 bytes in z/VSE).

Optional parameters

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE). Specifies the default for message-persistence of messages published to the topic. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be:
MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.
MQPER_PERSISTENT
Message is persistent.

There are only persistent messages in z/VSE.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY). Specifies the default priority of messages published to the topic. Values are ignored in z/VSE.

Specify either:
integer
The default priority to be used, in the range zero through to the maximum priority value that is supported (9).
MQPRI_PRIORITY_AS_PARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DefPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE). The value can be:
MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.
MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB). The value can be:
MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.
MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.
MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB). The value can be:
MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_PUB_INHIBITED
Publications are inhibited for this topic.
MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB). The value can be:
MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.
MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.
MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY). The value can be:
MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.
MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC). Text that briefly describes the object

The maximum length is MQ_TOPIC_DESC_LENGTH.

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

TopicType (MQCFIN)
Topic type (parameter identifier: MQIA_TOPIC_TYPE). The value specified must match the type of the topic being changed. The value can be:

MQTOPT_LOCAL
Local topic object

Clear Topic String

The Clear Topic String (MQCMD_CLEAR_TOPIC_STRING) command clears the retained message which is stored for the specified topic.

Required parameters:

TopicString, ClearType

Optional parameters:

Scope

Required parameters

TopicString (MQCFST)
Topic String (parameter identifier: MQCA_TOPIC_STRING). The topic string to be cleared The maximum length of the string is MQ_TOPIC_STR_LENGTH.

ClearType (MQCFIN)
Clear type (parameter identifier: MQIACF_CLEAR_TYPE). Specifies the type of clear command being issued.

The value must be:

MQCLRT_RETAINED
Remove the retained publication from the specified topic string.

Optional parameters

Scope (MQCFIN)
Scope of clearance (parameter identifier: MQIACF_CLEAR_SCOPE). Whether the topic string is to be cleared locally or globally. The value can be:

MQCLRS_LOCAL
The retained message is removed from the specified topic string at the local queue manager only.

Delete Topic

The Delete Topic (MQCMD_DELETE_TOPIC) command deletes the specified administrative topic object.

Required parameters:

TopicName

Optional parameters:

None

Required parameters

TopicName (MQCFST)
The name of the administrative topic definition to be deleted (parameter identifier: MQCA_TOPIC_NAME). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Inquire Topic

The Inquire Topic (MQCMD_INQUIRE_TOPIC) command inquires about the attributes of existing WebSphere MQ administrative topic objects.

Required parameters:

TopicName

Optional parameters:

IntegerFilterCommand, StringFilterCommand, TopicAttrs, TopicType

Required parameters

TopicName (MQCFST)
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object about which information is to be returned. Generic topic object names are supported. A generic name is a character string followed by an asterisk (*). For example, ABC* selects all administrative topic objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier: must be any integer type parameter allowed in TopicAttrs except MQIACF_ALL.

Use this parameter to restrict the output from the command by specifying a filter condition. If you specify an integer filter, you cannot also specify a string filter using the StringFilterCommand parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier: must be any string type parameter allowed in TopicAttrs except MQCA_TOPIC_NAME. Use this parameter to restrict the output from the command by specifying a filter condition.

If you specify a string filter, you cannot also specify an integer filter using the IntegerFilterCommand parameter.

TopicAttrs (MQCFIL)
Topic object attributes (parameter identifier: MQIACF_TOPIC_ATTRS). The attribute list can specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:
MQCA_ALTERATION_DATE
The date on which the information was last altered.
MQCA_ALTERATION_TIME
The time at which the information was last altered.
MQCA_MODEL_DURABLE_Q
Name of the model queue for durable managed subscriptions.
MQCA_MODEL_NON_DURABLE_Q
Name of the model queue for non-durable managed subscriptions.
MQCA_TOPIC_DESC
Description of the topic object.
MQCA_TOPIC_NAME
Name of the topic object.
MQCA_TOPIC_STRING
The topic string for the topic object.
MQIA_DEF_PRIORITY
Default message priority.
MQIA_DEF_PUT_RESPONSE_TYPE
Default put response.
MQIA_DURABLE_SUB
Whether durable subscriptions are permitted.
MQIA_INHIBIT_PUB
Whether publications are allowed.
MQIA_INHIBIT_SUB
Whether subscriptions are allowed.
MQIA_PM_DELIVERY
The delivery mechanism for persistent messages.

TopicType (MQCFIN)
Cluster information (parameter identifier: MQIA_TOPIC_TYPE).

If this parameter is present, eligible queues are limited to the specified type. Any attribute selector that is specified in the TopicAttrs list and that is valid only for topics of different type is ignored; no error is raised.

If this parameter is not present (or if MQIACF_ALL is specified), queues of all types are eligible. Each attribute specified must be a valid topic attribute selector (that is, it must be in the following list), but it need not be applicable to all or any of the topics returned. Topic attribute selectors that are valid but not applicable to the queue are ignored; no error messages occur and no attribute is returned.

The value can be:

MQTOPT_ALL
All topic types are displayed. MQTOPT_ALL is the default value.

MQTOPT_LOCAL
Locally defined topics are displayed.

Inquire Topic Names

The Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command returns a list of administrative topic names that match the generic topic name specified.

Required parameters:

TopicName

Optional parameters:

None

Required parameters

TopicName (MQCFST)
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object that information is to be returned for.

Generic topic object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Inquire Topic Status

The Inquire Topic Status (MQCMD_INQUIRE_TOPIC_STATUS) command inquires the status of a particular topic, or of a topic and its child topics.

Required parameters:

TopicString

Optional parameters:

StatusType, IntegerFilterCommand, StringFilterCommand, TopicStatusAttrs

Required parameters

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING). In WebSphere MQ for z/VSE, only the trailing wildcard of # is allowed.

The maximum length of the string is 256.

Optional parameters

StatusType (MQCFIN)
The type of status to return (parameter identifier: MQIACF_TOPIC_STATUS_TYPE).

The value can be:
- MQIACF_TOPIC_STATUS
- MQIACF_TOPIC_SUB
- MQIACF_TOPIC_PUB

This command ignores any attribute selectors specified in the TopicStatusAttrs list that are not valid for the selected StatusType and the command raises no error.

The default value if this parameter is not specified is MQIACF_TOPIC_STATUS.

IntegerFilterCommand(MQCFIF)
Integer filter command descriptor that you use to restrict the output from the command. The parameter identifier: must be an integer type and must be one of the values allowed for MQIACF_TOPIC_SUB_STATUS, MQIACF_TOPIC_PUB_STATUS or MQIACF_TOPIC_STATUS, except MQIACF_ALL.

If you specify an integer filter, you cannot also specify a string filter with the StringFilterCommand parameter.

StringFilterCommand(MQCFSF)
String filter command descriptor. The parameter identifier: must be any string type parameter allowed for MQIACF_TOPIC_SUB_STATUS, MQIACF_TOPIC_PUB_STATUS or MQIACF_TOPIC_STATUS, except MQIACF_ALL, or the identifier MQCA_TOPIC_STRING_FILTER to filter on the topic string.

Use the parameter identifier to restrict the output from the command by specifying a filter condition. Ensure that the parameter is valid for the type selected in StatusType. If you specify a string filter, you cannot also specify an integer filter using the IntegerFilterCommand parameter.

TopicStatusAttrs(MQCFIL)
Topic status attributes (parameter identifier: MQIACF_TOPIC_STATUS_ATTRS)

The default value used if the parameter is not specified is: MQIACF_ALL

You can specify any of the parameter values listed in the related reference about Response Data. It is not an error to request status information that is not relevant for a particular status type, but the response contains no information for the value concerned.

Data responses to commands

Escape commands, and commands that request information, if successful, generate data responses. A data response consists of an OK response (as described in "OK response" on page 240) followed by additional structures containing the requested data.

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

Data responses are generated for these commands:

- Escape
- Inquire Channel Authentication Records
- Inquire Channel Names
- Inquire Channel Status
- Inquire Channel
- Inquire Namelist Names
- Inquire Namelist
- Inquire Queue Manager
- Inquire Queue Names
- Inquire Queue
- Inquire Subscription Status
- Inquire Subscription
- Inquire Topic
- Inquire Topic Names
- Inquire Topic Status

Escape (Response)

The response to the Escape (MQCMD_ESCAPE) command consists of the response header followed by two parameter structures, one containing the escape type, and the other containing the text response. More than one such message may be issued, depending upon the command contained in the Escape request.

The Command field in the response header MQCFH contains the MQCMD_* command identifier of the text command contained in the EscapeText parameter in the original Escape command. For example, if EscapeText in the original Escape command specified PING QMGR, Command in the response has the value MQCMD_PING_Q_MGR.

If it is possible to determine the outcome of the command, the CompCode in the response header identifies whether the command was successful. The success or otherwise can therefore be determined without the recipient of the response having to parse the text of the response.

If it is not possible to determine the outcome of the command, CompCode in the response header has the value MQCC_UNKNOWN, and Reason is MQRC_NONE.

Always returned:

EscapeType
EscapeText

Returned if requested:

None

Parameters:

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
WebSphere MQ command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string holding the response to the original command.

Inquire Channel (Response)

The response to the Inquire Channel (MQCMD_INQUIRE_CHANNEL) command consists of the response header followed by the ChannelName structure and the requested combination of attribute parameter structures (where applicable). If a generic channel name was specified, one such message is generated for each channel found.

This response is supported on all platforms.

Always returned:
ChannelName

Returned if requested:
AlterationDate, AlterationTime, BatchSize, BatchInterval, ChannelDesc, ChannelMonitoring, ChannelStatistics, ChannelType, ConnectionName, DataConversion, DiscInterval, DiscRetryCount, LongRetryCount, LongRetryInterval, MaxMsgLength, MsgExit, MsgUserData, PortNumber, PropertyControl, PropertyControl, ReceiveExit, ReceiveUserData, SecurityExit, SecurityUserData, SendExit, SendUserData, SeqNumberWrap, ShortRetryCount, ShortRetryInterval, SSLCipherSpec, SSLClientAuth, SSLPeerName, TpName, TransportType, XmitQName

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's ChannelMonitoring parameter is inherited by the channel. This is the default value.

MQMON_LOW
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's ChannelMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStatistics (MQCFIN)
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL). Specifies whether statistics data is to be collected and, if so, the rate at which the data is collected.

The value can be:

MQMON_OFF
Statistics data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's ChannelStatistics parameter is inherited by the channel.

MQMON_LOW
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's ChannelStatistics parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value may be:

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_SVRCONN
Server-connection (for use by clients).

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

DiscRetryCount (MQCFIN)
Disconnection retry count (parameter identifier: MQIACH_DISC_RETRY).

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. If the channel has a chain of exits, a list of names is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 exit names in the MQCFSL structure, each with a length of MQ_EXIT_NAME_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the user data depends on the environment in which the exit is running. MQ_EXIT_DATA_LENGTH gives the maximum length for the environment in which your application is running. If the

channel has a chain of exits, a list of data values is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 user data values in the MQCFSL structure, each with a length of MQ_EXIT_DATA_LENGTH.

PortNumber (MQCFIN)
TCP/IP port number (parameter identifier: MQIACH_PORT_NUMBER).

PropertyControl (MQCFIN)
Message property control. Specifies what happens to properties of messages when the message is about to be sent to a queue manager that does not understand the concept of a property descriptor. This parameter is applicable to Sender and Server channels. The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise, all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. If the channel has a chain of exits, a list of names is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 exit names in the MQCFSL structure, each with a length of MQ_EXIT_NAME_LENGTH.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the user data depends on the environment in which the exit is running. MQ_EXIT_DATA_LENGTH gives the maximum length for the environment in which your application is running. If the channel has a chain of exits, a list of data values is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 user data values in the MQCFSL structure, each with a length of MQ_EXIT_DATA_LENGTH.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. If the channel has a chain of exits, a list of names is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 exit names in the MQCFSL structure, each with a length of MQ_EXIT_NAME_LENGTH.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the user data depends on the environment in which the exit is running. MQ_EXIT_DATA_LENGTH gives the maximum length for the environment in which your application is running. If the channel has a chain of exits, a list of data values is returned in an MQCFSL structure instead of an MQCFST structure.

WebSphere MQ for z/VSE can return up to 8 user data values in the MQCFSL structure, each with a length of MQ_EXIT_DATA_LENGTH.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLCipherSpec (MQCFIN)
SSL cipher specification (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

SSLClientAuth (MQCFIN)
SSL client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_OPTIONAL
Client authentication is required.

MQSCA_REQUIRED
Client authentication is optional.

SSLPeerName (MQCFST)
SSL peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The maximum length of the string is MQ_DISTINGUISHED_NAME_LENGTH.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Channel Listener (Response)

The response to the Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command consists of the response header followed by the ListenerName structure and the requested combination of attribute parameter structures. If a generic listener name was specified, one such message is generated for each listener found.

Always returned:

ListenerName

Returned if requested:

AlterationDate, AlterationTime, Backlog, IPAddress, ListenerDesc, Port, StartMode, TransportType

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form *yyyy-mm-dd*, on which the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form *hh.mm.ss*, at which the information was last altered.

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName (MQCFST)
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME)

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. This is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

TransportType (MQCFIN)
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE). The value can be:

MQXPT_TCP TCP

Inquire Channel Listener Status (Response)

The response to the Inquire Channel Listener Status command (MQCMD_INQUIRE_LISTENER_STATUS) consists of the response header followed by the ListenerName structure and the requested combination of attribute parameter structures. If a generic listener name was specified, one such message is generated for each listener found. Only listeners with status of RUNNING will be returned a status response.

Always returned:

ListenerName

Returned if requested:

Backlog, IPAddress, ListenerDesc, Port, StartDate, StartMode, StartTime, Status, TransportType

Response data

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName (MQCFST)
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP.

StartDate (MQCFST)
Start date (parameter identifier: MQCACH_LISTENER_START_DATE).

The date, in the form *yyyy-mm-dd*, on which the listener was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. This is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

StartTime (MQCFST)
Start time (parameter identifier: MQCACH_LISTENER_START_TIME).

The time, in the form *hh.mm.ss*, at which the listener was started.

The maximum length of the string is MQ_TIME_LENGTH

Status (MQCFIN)
Listener status (parameter identifier: MQIACH_LISTENER_STATUS).

The current status of the listener. The value can be:
MQSVC_STATUS_RUNNING
The listener is running.
MQSVC_STATUS_STOPPED
The listener is stopped.

TransportType (MQCFIN)
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE). The value can be:

MQXPT_TCP TCP

ProcessId (MQCFIN)
The listener CICS task number (parameter identifier: MQIACF_PROCESS_ID).

Inquire Channel Names (Response)

The response to the Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified channel name.

This response is supported on all platforms.

Always returned:
ChannelNames

Returned if requested:
None

Response data

ChannelNames (MQCFSL)
Channel names (parameter identifier: MQCACH_CHANNEL_NAMES).

Inquire Channel Status (Response)

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by:
- The ChannelName structure
- The ChannelInstanceType structure
- The ChannelStatus structure
- The ChannelType structure
- The ConnectionName structure
- The RemoteQMgrName structure
- The StopRequested structure
- The XmitQName structure

which are followed by the requested combination of status attribute parameter structures. One such message is generated for each channel instance found that matches the criteria specified on the command.

Always returned:

ChannelInstanceType, ChannelName, ChannelStatus, ChannelType, ConnectionName, RemoteQMgrName, StopRequested, SubState, XmitQName

Returned if requested:

Batches, BatchSize, BatchSizeIndicator, BuffersReceived, BuffersSent, BytesReceived, BytesSent, ChannelMonitoring, ChannelStartDate, ChannelStartTime, CurrentLUWID, CurrentMsgs, CurrentSequenceNumber, CurrentSharingConversations, ExitTime, InDoubtStatus, LastLUWID, LastMsgDate, LastMsgTime, LastSequenceNumber, LocalAddress, LongRetriesLeft, MaxSharingConversations, MCAStatus, MCAUserIdentifier, Msgs, MsgsAvailable,

NetTime, QMgrName, ShortRetriesLeft, SSLCertRemoteIssuerName, SSLCertUserId, SSLKeyResetDate, SSLKeyResets, SSLKeyResetTime, SSLShortPeerName, XQTime

Response data

Batches (MQCFIN)
Number of completed batches (parameter identifier: MQIACH_BATCHES).

BatchSize (MQCFIN)
Negotiated batch size (parameter identifier: MQIACH_BATCH_SIZE).

BatchSizeIndicator (MQCFIL)
Indicator of the number of messages in a batch (parameter identifier: MQIACH_BATCH_SIZE_INDICATOR). Two values are returned:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

BuffersReceived (MQCFIN)
Number of buffers received (parameter identifier: MQIACH_BUFFERS_RCVD).

BuffersSent (MQCFIN)
Number of buffers sent (parameter identifier: MQIACH_BUFFERS_SENT).

BytesReceived (MQCFIN)
Number of bytes received (parameter identifier: MQIACH_BYTES_RCVD).

BytesSent (MQCFIN)
Number of bytes sent (parameter identifier: MQIACH_BYTES_SENT).

ChannelInstanceType (MQCFIN)
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

The value can be:
MQOT_CURRENT_CHANNEL
Current channel status.
MQOT_SAVED_CHANNEL
Saved channel status.

ChannelMonitoring (MQCFIN)
Current level of monitoring data collection for the channel (parameter identifier: MQIACH_MONITORING_CHANNEL).

The value can be:
MQMON_OFF
Monitoring for the channel is switched off.
MQMON_LOW
Low rate of data collection.
MQMON_MEDIUM
Medium rate of data collection.
MQMON_HIGH
High rate of data collection.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStartDate (MQCFST)
Date channel started, in the form yyyy-mm-dd (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

ChannelStartTime (MQCFST)
Time channel started, in the form hh.mm.ss (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS). The value can be:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_INITIALIZING
Channel is initializing.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE). The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

CurrentLUWID (MQCFST)
Logical unit of work identifier for in-doubt batch (parameter identifier: MQCACH_CURRENT_LUWID).

The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in-doubt it is the LUWID of the in-doubt batch.

It is updated with the LUWID of the next batch when this is known.

The maximum length is MQ_LUWID_LENGTH.

CurrentMsgs (MQCFIN)
Number of messages in-doubt (parameter identifier: MQIACH_CURRENT_MSGS). For a sending channel, this is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in-doubt it is the number of messages that are in-doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

CurrentSequenceNumber (MQCFIN)
Sequence number of last message in in-doubt batch (parameter identifier: MQIACH_CURRENT_SEQ_NUMBER).

For a sending channel, this is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in-doubt it is the message sequence number of the last message in the in-doubt batch.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

CurrentSharingConversations (MQCFIN)
Number of conversations currently active on this channel instance (parameter identifier: MQIACH_CURRENT_SHARING_CONVS).

This is returned only for TCP/IP server-connection channels.

ExitTime (MQCFIL)
Indicator of the time taken executing user exits per message (parameter identifier: MQIACH_EXIT_TIME_INDICATOR). Amount of time, in microseconds, spent processing user exits per message. Where more than one exit is executed per message, the value is the sum of all the user exit times for a single message. Two values are returned:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

InDoubtStatus (MQCFIN)
Whether the channel is currently in doubt (parameter identifier: MQIACH_INDOUBT_STATUS).

A sending channel is only in doubt while the sending Message Channel Agent is waiting for an acknowledgement that a batch of messages, which it has sent, has been successfully received. It is not in doubt at all other times, including the period during which messages are being sent, but before an acknowledgement has been requested.

A receiving channel is never in doubt.

The value can be:

MQCHIDS_NOT_INDOUBT
Channel is not in-doubt.
MQCHIDS_INDOUBT
Channel is in-doubt.

LastLUWID (MQCFST)
Logical unit of work identifier for last committed batch (parameter identifier: MQCACH_LAST_LUWID).

The maximum length is MQ_LUWID_LENGTH.

LastMsgDate (MQCFST)
Date last message was sent, or MQI call was handled, in the form yyyy-mm-dd (parameter identifier: MQCACH_LAST_MSG_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

LastMsgTime (MQCFST)
Time last message was sent, or MQI call was handled, in the form hh.mm.ss (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

LastSequenceNumber (MQCFIN)
Sequence number of last message in last committed batch (parameter identifier: MQIACH_LAST_SEQ_NUMBER).

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetriesLeft (MQCFIN)
Number of long retry attempts remaining (parameter identifier: MQIACH_LONG_RETRIES_LEFT).

MaxSharingConversations (MQCFIN)
Maximum number of conversations permitted on this channel instance. (parameter identifier: MQIACH_MAX_SHARING_CONVS)

This is returned only for TCP/IP server-connection channels.

MCAStatus (MQCFIN)
MCA status (parameter identifier: MQIACH_MCA_STATUS). The value can be:
MQMCAS_STOPPED
Message channel agent stopped.
MQMCAS_RUNNING
Message channel agent running.

MCAUserIdentifier (MQCFST)
The user ID used by the MCA (parameter identifier: MQCACH_MCA_USER_ID). This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

Msgs (MQCFIN)
Number of messages sent or received, or number of MQI calls handled (parameter identifier: MQIACH_MSGS).

MsgsAvailable (MQCFIN)
Number of messages available (parameter identifier:

MQIACH_XMITQ_MSGS_AVAILABLE). Number of messages queued on the transmission queue available to the channel for MQGETs.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

NetTime (MQCFIL)
Indicator of the time of a network operation (parameter identifier: MQIACH_NETWORK_TIME_INDICATOR). Amount of time, in microseconds, to send a request to the remote end of the channel and receive a response. Two values are returned:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

QMgrName (MQCFST)
Name of the queue manager that owns the channel instance (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of the remote queue manager, or queue-sharing group (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

ShortRetriesLeft (MQCFIN)
Number of short retry attempts remaining (parameter identifier: MQIACH_SHORT_RETRIES_LEFT).

SSLCertRemoteIssuerName (MQCFST)
The full Distinguished Name of the issuer of the remote certificate. The issuer is the Certificate Authority that issued the certificate (parameter identifier: MQCACH_SSL_CERT_ISSUER_NAME).

The maximum length of the string is MQ_SHORT_DNAME_LENGTH.

SSLKeyResetDate (MQCFST)
Date of the previous successful SSL secret key reset, in the form yyyy-mm-dd (parameter identifier: MQCACH_SSL_KEY_RESET_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

SSLKeyResets (MQCFIN)
SSL secret key resets (parameter identifier: MQIACH_SSL_KEY_RESETS). The number of successful SSL secret key resets that have occurred for this channel instance since the channel started. If SSL secret key negotiation is enabled, the count is incremented whenever a secret key reset is performed.

SSLKeyResetTime (MQCFST)
Time of the previous successful SSL secret key reset, in the form hh.mm.ss (parameter identifier: MQCACH_SSL_KEY_RESET_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

SSLShortPeerName (MQCFST)
Distinguished Name of the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_SHORT_PEER_NAME).

The maximum length is MQ_SHORT_DNAME_LENGTH. This limit might mean that exceptionally long Distinguished Names are truncated.

StopRequested (MQCFIN)
Whether user stop request is outstanding (parameter identifier: MQIACH_STOP_REQUESTED).

The value can be:
MQCHSR_STOP_NOT_REQUESTED
User stop request has not been received.
MQCHSR_STOP_REQUESTED
User stop request has been received.

SubState (MQCFIN)
Current action being performed by the channel (parameter identifier: MQIACH_CHANNEL_SUBSTATE).

The value can be:
MQCHSSTATE_CHADEXIT
Running channel auto-definition exit.
MQCHSSTATE_COMPRESSING
Compressing or decompressing data.
MQCHSSTATE_END_OF_BATCH
End of batch processing.
MQCHSSTATE_HANDSHAKING
SSL handshaking.
MQCHSSTATE_HEARTBEATING
Heartbeating with partner.
MQCHSSTATE_IN_MQGET
Performing MQGET.
MQCHSSTATE_IN_MQI_CALL
Executing an MQ API call, other than an MQPUT or MQGET.
MQCHSSTATE_IN_MQPUT
Performing MQPUT.
MQCHSSTATE_MREXIT
Running retry exit.
MQCHSSTATE_MSGEXIT
Running message exit.
MQCHSSTATE_NAME_SERVER
Nameserver request.
MQCHSSTATE_NET_CONNECTING
Network connect.
MQCHSSTATE_OTHER
Undefined state.
MQCHSSTATE_RCVEXIT
Running receive exit.
MQCHSSTATE_RECEIVING
Network receive.
MQCHSSTATE_RESYNCHING
Resynching with partner.
MQCHSSTATE_SCYEXIT
Running security exit.
MQCHSSTATE_SENDEXIT
Running send exit.
MQCHSSTATE_SENDING
Network send.
MQCHSSTATE_SERIALIZING
Serialized on queue manager access.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XQTime (MQCFIL)
Transmission queue time indicator (parameter identifier: MQIACH_XMITQ_TIME_INDICATOR). The time, in microseconds, that messages remained on the transmission queue before being retrieved. The time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application.

Two values are returned:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

Inquire Connection (Response)

The response to the Inquire Connection (MQCMD_INQUIRE_CONNECTION) command consists of the response header followed by the ConnectionId structure and a set of attribute parameter structures determined by the value of ConnInfoType in the Inquire command.

If the value of ConnInfoType was MQIACF_CONN_INFO_ALL, there is one message for each connection found with MQIACF_CONN_INFO_CONN, and n more messages per connection with MQIACF_CONN_INFO_HANDLE (where n is the number of objects that the connection has open).

Always returned:
ConnectionId, ConnInfoType

Always returned if ConnInfoType is MQIACF_CONN_INFO_HANDLE:
ObjectName, ObjectType

Returned if requested and ConnInfoType is MQIACF_CONN_INFO_CONN:
ApplTag, ApplType, ChannelName, ConnectionName, ConnectionOptions, TaskNumber, TransactionId, UOWStartDate, UOWStartTime, UOWState, UOWType, UserId

Returned if requested and ConnInfoType is MQIACF_CONN_INFO_HANDLE:
HandleState, OpenOptions

Response data

ApplTag (MQCFST)
Application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length is MQ_APPL_TAG_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

The value can be: MQAT_BATCH Application using a batch connection. MQAT_CICS CICS transaction.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ConnectionId (MQCFBS)
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

The length of the string is MQ_CONNECTION_ID_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ConnectionOptions (MQCFIL)
Connect options currently in force for the connection (parameter identifier: MQIACF_CONNECT_OPTIONS).

ConnInfoType (MQCFIN)
Type of information returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value may be:
MQIACF_CONN_INFO_CONN
Generic information for the specified connection.
MQIACF_CONN_INFO_HANDLE
Information pertinent only to those objects opened by the specified connection.

HandleState (MQCFIN)
State of the handle (parameter identifier: MQIACF_HANDLE_STATE).

The value may be:
MQHSTATE_ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.
MQHSTATE_INACTIVE
No API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

ObjectName (MQCFST)
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ObjectType (MQCFIN)
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:
MQOT_Q
Queue.
MQOT_NAMELIST
Namelist.
MQOT_Q_MGR
Queue manager.

OpenOptions (MQCFIN)
Open options currently in force for the object for connection (parameter identifier: MQIACF_OPEN_OPTIONS).

TaskNumber (MQCFST)
Task number (parameter identifier: MQCACF_TASK_NUMBER).

The CICS task number as a 7-digit string.

The maximum length of the string is MQ_TASK_NUMBER_LENGTH.

TransactionId (MQCFST)
Transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID).

The 4-character CICS transaction identifier.

The maximum length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWStartDate (MQCFST)
Unit of work creation date (parameter identifier: MQCACF_UOW_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

UOWStartTime (MQCFST)
Unit of work creation time (parameter identifier: MQCACF_UOW_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

UOWState (MQCFIN)
State of the unit of work (parameter identifier: MQIACF_UOW_STATE).

The value can be:
MQUOWST_NONE
There is no unit of work.
MQUOWST_ACTIVE
The unit of work is active.

UOWType (MQCFIN)
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be:

MQUOWT_CICS
CICS.

UserId (MQCFST)
User identifier (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Namelist (Response)

The response to the Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command consists of the response header followed by the NamelistName structure and the requested combination of attribute parameter structures (where applicable).

If a generic namelist name was specified, one such message is generated for each namelist found.

This response is supported on all platforms.

Always returned:
NamelistName

Returned if requested:

AlterationDate, AlterationTime, NameCount, NamelistDesc, Names

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

NameCount (MQCFIN)
Number of Names associated with the namelist (parameter identifier: MQIA_NAME_COUNT).

NamelistDesc (MQCFST)
Namelist description (parameter identifier: MQCA_NAMELIST_DESC).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Names (MQCFSL)
Namelist names. These are the names associated with the namelist object (parameter identifier: MQCA_NAMES).

The NameCount attribute indicates how many names are associated with the namelist object, and the MQCFSL structure StringLength field specifies the length of each name.

Inquire Namelist Names (Response)

The response to the Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified namelist name.

This response is supported on all platforms.

Always returned:

NamelistNames

Returned if requested:

None

Response data

NamelistNames (MQCFSL) Namelist names (parameter identifier: MQCACF_NAMELIST_NAMES).

Inquire Queue (Response)

The response to the Inquire Queue (MQCMD_INQUIRE_Q) command consists of the response header followed by the QName structure and the requested

combination of attribute parameter structures. If a generic queue name was specified, one such message is generated for each queue found.

This PCF is supported on all platforms.

Always returned:

QName

Returned if requested:

AlterationDate, AlterationTime, BaseQName, CICSFileName, CreationDate, CreationTime, DefinitionType, DefPersistence, InhibitGet, InhibitPut, MaxGlobalLocks, MaxLocalLocks, MaxMsgLength, MaxQDepth, MaxQTriggers, MaxQUsers, PropertyControl, QDepthHighEvent, QDepthHighLimit, QDepthLowEvent, QDepthLowLimit, QDepthMaxEvent, QDesc, QServiceInterval, QServiceIntervalEvent, QType, QueueAccounting, QueueMonitoring, QueueStatistics, RemoteQMgrName, RemoteQName, Shareability, TriggerChannelName, TriggerControl, TriggerData, TriggerProgramName, TriggerRestart, TriggerTerminalId, TriggerTransactionId, TriggerType, Usage, XmitQName

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CICSFileName (MQCFST)
CSD file name for queue messages (parameter identifier: MQCA_CICS_FILE_NAME).

The maximum length of the string is MQ_CICS_FILE_NAME_LENGTH.

CreationDate (MQCFST)
Queue creation date (parameter identifier: MQCA_CREATION_DATE).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime (MQCFST)
Creation time (parameter identifier: MQCA_CREATION_TIME).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE). The value can be:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE). Specifies the default for message-persistence on the queue.

Message persistence determines whether or not messages are preserved across restarts of the queue manager.

The value can be:

MQPER_PERSISTENT
Default message persistence is persistent.

MQPER_NOT_PERSISTENT
Default message persistence is not persistent.

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

MaxGlobalLocks (MQCFIN)
Buffer size for queue manager to manage concurrent queue access (parameter identifier: MQIA_MAX_GLOBAL_LOCKS).

MaxLocalLocks (MQCFIN)
Buffer size for applications to manage concurrent queue access (parameter identifier: MQIA_MAX_LOCAL_LOCKS).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MaxQTriggers (MQCFIN)
Maximum number of concurrent trigger instances for a particular queue (parameter identifier: MQIA_MAX_Q_TRIGGERS).

MaxQUsers (MQCFIN)
Maximum number of active opens to any particular queue (parameter identifier: MQIA_Q_USERS).

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

The value can be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

MQQSIE_OK
Queue Service Interval OK events enabled.

MQQSIE_NONE
No queue service interval events enabled.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_MODEL
Model queue definition.

MQQT_REMOTE
Local definition of a remote queue.

QueueAccounting (MQCFIN)
Controls the collection of accounting data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:
MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the QueueAccounting parameter on the queue manager.
MQMON_OFF
Accounting data collection is disabled for the queue.
MQMON_ON
If the value of the queue manager's QueueAccounting parameter is not MQMON_NONE, accounting data collection is enabled for the queue.

QueueMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be:
MQMON_OFF
Online monitoring data collection is turned offfor this queue.

MQMON_Q_MGR
The value of the queue manager's QueueMonitoring parameter is inherited by the queue. This is the default value.

MQMON_LOW
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this queue.

MQMON_MEDIUM
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this queue.

MQMON_HIGH
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this queue.

QueueStatistics (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_STATISTICS_Q). Specifies whether statistics data collection is enabled.

The value can be:

MQMON_Q_MGR
The value of the queue manager's QueueStatistics parameter is inherited by the queue.

MQMON_OFF
Statistics data collection is disabled

MQMON_ON
If the value of the queue manager's QueueMonitoring parameter is not MQMON_NONE, statistics data collection is enabled.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

TriggerChannelName (MQCFST)
Channel name for MCA trigger process (parameter identifier: MQCA_TRIGGER_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

The value may be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

TriggerProgramName (MQCFST)
Program name for trigger process (parameter identifier: MQCA_TRIGGER_PROGRAM_NAME).

The maximum length of the string is MQ_TRIGGER_PROGRAM_NAME_LENGTH.

TriggerRestart (MQCFIN)
Indicator for the reactivation of a trigger process (parameter identifier: MQIA_TRIGGER_RESTART).

The value can be:

MQTRIGGER_RESTART_NO
Do not reactivate trigger process.

MQTRIGGER_RESTART_YES
Reactivate trigger process.

TriggerTerminalId (MQCFST)
Terminal identifier for trigger process (parameter identifier: MQCA_TRIGGER_TERM_ID).

The maximum length of the string is MQ_TRIGGER_TERM_ID_LENGTH.

TriggerTransactionId (MQCFST)
Transaction identifier for trigger process (parameter identifier: MQCA_TRIGGER_TRANS_ID).

The maximum length of the string is MQ_TRIGGER_TRANS_ID_LENGTH.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value may be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Queue Manager (Response)

The response to the Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command consists of the response header followed by the QMgrName structure and the requested combination of attribute parameter structures.

This response is supported on all platforms.

Always returned:

QMgrName

Returned if requested:

AccountingConnOverride, AccountingInterval, AlterationDate, AlterationTime, AuthorityEvent, BatchInterfaceAutoStart, BatchInterfaceId, ChannelAuthenticationRecords, ChannelAutoDef, ChannelAutoDefEvent, ChannelAutoDefExit, ChannelMonitoring, ChannelStatistics, CodedCharSetId, CommandInputQName, CommandEvent, CommandLevel, CommandReplyQName, CommandServerAutoStart, CommandServerDataConversion, CommandServerDeadLetterQ, ConfigurationEvent, DeadLetterQName, DistLists, InhibitEvent, ListenerPortNumber, LocalEvent, MaxGlobalLocks, MaxHandles, MaxPropertiesLength, MaxLocalLocks, MaxMsgLength, MaxOpenQ, MaxQDepth, MaxQUsers, MonitorInterval, MonitorQName, MQIAccounting, MQIStatistics, PerformanceEvent, Platform, QMgrDesc, QueueAccounting, QueueMonitoring, QueueStatistics, RemoteEvent, SSLEvent, SSLKeyLibraryMember, SSLKeyLibraryName, SSLResetCount, StartStopEvent, StatisticsInterval, SyncPoint, SystemLogQName

Response data

AccountingConnOverride (MQCFIN)
Specifies whether applications can override the settings of the QueueAccounting and MQIAccounting queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be:

MQMON_DISABLED
Applications cannot override the settings of the QueueAccounting and MQIAccounting parameters. This is the queue manager's initial default value.

MQMON_ENABLED
Applications can override the settings of the QueueAccounting and MQIAccounting parameters by using the options field of the MQCNO structure of the MQCONNX API call.

AccountingInterval (MQCFIN)
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

Specify a value in the range 1 through 604 000.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

BatchInterfaceAutoStart (MQCFIN)
Indicator for the automatic activation of the batch interface (parameter identifier: MQIA_BATCH_INTERFACE_AUTO).

The value can be:

MQAUTO_START_NO
Do not automatically start the batch interface.

MQAUTO_START_YES
Automatically start the batch interface.

BatchInterfaceId (MQCFST)
Batch interface identifier (parameter identifier: MQCA_BATCH_INTERFACE_ID).

The maximum length of the string is MQ_BATCH_INTERFACE_ID_LENGTH.

ChannelAuthenticationRecords (MQCFIN)
Controls whether channel authentication records are checked (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:
MQCHLA_DISABLED
Channel authentication records are not checked.
MQCHLA_ENABLED
Channel authentication records are checked.

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent (MQCFIN)
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver or server-connection channel is auto-defined.

The value can be:

MQEVR_DISABLED
Event reporting disabled

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ChannelMonitoring (MQCFIN)
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:
MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their ChannelMonitoring parameter.
MQMON_OFF
Online monitoring data collection is turned off for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter. This is the queue manager's initial default value.
MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.
MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.
MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelMonitoring parameter.

ChannelStatistics (MQCFIN)
Controls whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:
MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their ChannelStatistics parameter. This is the queue manager's initial default value.

MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.
MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.
MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.
MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their ChannelStatistics parameter.

CodedCharSetId (MQCFIN)
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:
MQEVR_DISABLED
Event reporting disabled.
MQEVR_ENABLED
Event reporting enabled.
MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandInputQName (MQCFST)
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandLevel (MQCFIN)
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

For WebSphere MQ for z/VSE, the value is MQCMDL_LEVEL_600.

CommandReplyQName (MQCFST)
MQSC reply queue name (parameter identifier: MQCA_COMMAND_REPLY_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandServerAutoStart (MQCFIN)
Indicator for the automatic activation of the PCF command server (parameter identifier: MQIA_CMD_SERVER_AUTO).

The value can be:

MQAUTO_START_NO
Do not automatically start the PCF command server.

MQAUTO_START_YES
Automatically start the PCF command server.

CommandServerDataConversion (MQCFIN)
Indicator for the data conversion of PCF messages (parameter identifier MQIA_CMD_SERVER_CONVERT_MSG).

The value can be:

MQCSRV_CONVERT_NO
Do not convert PCF messages.

MQCSRV_CONVERT_YES
Convert PCF messages.

CommandServerDeadLetterQ (MQCFIN)
Indicator for the storage of undeliverable PCF reply messages (parameter identifier: MQIA_CMD_SERVER_DLQ_MSG).

The value bay be:

MQCSRV_DLQ_NO
Do not store undeliverable PCF replies to DLQ.

MQCSRV_DLQ_YES
Store undeliverable PCF replies to DLQ.

ConfigurationEvent (MQCFIN)
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

For WebSphere MQ for z/VSE, the value is MQDL_NOT_SUPPORTED.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ListenerPortNumber (MQCFIN)
Port number for TCP/IP Listener process (parameter identifier: MQIA_LISTENER_PORT_NUMBER).

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MaxGlobalLocks (MQCFIN)
Buffer size for queue manager to manage concurrent queue access (parameter identifier: MQIA_MAX_GLOBAL_LOCKS).

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

Specifies the maximum number of MQI connections that will be handled by the queue manager at any one time.

The value may be in the range 1 through 1000.

MaxLocalLocks (MQCFIN)
Buffer size for applications to manage concurrent queue access (parameter identifier: MQIA_MAX_LOCAL_LOCKS).

MaxPropertiesLength (MQCFIN)
Maximum priorities length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxOpenQ (MQCFIN)
Maximum number of concurrently open queues (parameter identifier: MQIA_MAX_OPEN_Q).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MaxQUsers (MQCFIN)
Maximum number of active opens to any particular queue (parameter identifier: MQIA_Q_USERS).

MonitorInterval (MQCFIN)
Queue manager housekeeping process interval (parameter identifier: MQIA_MONITOR_INTERVAL).

MonitorQName (MQCFST)
MQI monitor queue name (parameter identifier: MQCA_MONITOR_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

MQIAccounting (MQCFIN)
Controls whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled. This is the queue manager's initial default value.
MQMON_ON
MQI accounting data collection is enabled.

MQIStatistics (MQCFIN)
Controls whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:
MQMON_OFF
Data collection for MQI statistics is disabled. This is the queue manager's initial default value.
MQMON_ON
Data collection for MQI statistics is enabled.

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Platform (MQCFIN)
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

For WebSphere MQ for z/VSE, the value is PL_VSE.

QmgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QueueAccounting (MQCFIN)
Controls the collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:
MQMON_NONE
Accounting data collection for queues is disabled. This may not be overridden by the value of the QueueAccounting parameter on the queue.
MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the QueueAccounting parameter.
MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the QueueAccounting parameter.

QueueMonitoring (MQCFIN)
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the QueueMonitoring queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off. This is the queue manager's initial default value.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their QueueMonitoring attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

QueueStatistics (MQCFIN)
Controls whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their QueueStatistics parameter. This is the queue manager's initial default value.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their QueueStatistics parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their QueueStatistics parameter.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLEvent (MQCFIN)
Controls whether SSL events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLKeyLibraryMember (MQCFST)
SSL key library member name (parameter identifier: MQCA_SSL_KEY_MEMBER).

The maximum length of the string is MQ_SSL_KEY_MEMBER_LENGTH.

SSLKeyLibraryName (MQCFST)
SSL key library name (parameter identifier: MQCA_SSL_KEY_LIBRARY).

The maximum length of the string is MQ_SSL_KEY_LIBRARY_LENGTH.

SSLKeyResetCount (MQCFIN)
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

Specifies when SSL channel MCAs that initiate communication reset the secret key used for encryption on the channel. The value of this parameter represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. This number of bytes includes control information sent by the MCA. The secret key is renegotiated when (whichever occurs first): The total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or, If channel heartbeats are enabled, before data is sent or received following a channel heartbeat.

Specify a value in the range zero through 999 999 999. A value of zero, the queue manager's initial default value, signifies that secret keys are never renegotiated.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StatisticsInterval (MQCFIN)
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

Specify a value in the range 1 through 604 000.

SyncPoint (MQCFIN)
Syncpoint availability (parameter identifier: MQIA_SYNCPOINT).

For WebSphere MQ for z/VSE, the value is MQSP_NOT_AVAILABLE.

SystemLogQName
System log queue name (parameter identifier: MQCA_SYSTEM_LOG_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Queue Names (Response)

The response to the Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified queue name.

This response is supported on all platforms.

Always returned:

QNames

Returned if requested:

None

Response data

QNames (MQCFSL)
Queue names (parameter identifier: MQCACF_Q_NAMES).

Inquire Queue Status (Response)

The response to the Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command consists of the response header followed by the QName structure and a set of attribute parameter structures determined by the value of StatusType in the Inquire command.

Always returned:

QName, StatusType

Possible values of StatusType are:

MQIACF_Q_STATUS
Returns status information relating to queues.

MQIACF_Q_HANDLE
Returns status information relating to the handles that are accessing the queues.

Returned if requested and StatusType is MQIACF_Q_STATUS:

CurrentQDepth, LastGetDate, LastGetTime, LastPutDate, LastPutTime, OldestMsgAge, OnQTime, OpenInputCount, OpenOutputCount, QueueMonitoring, UncommittedMsgs

Returned if requested and StatusType is MQIACF_Q_HANDLE:

ApplTag, ApplType, ChannelName, ConnectionName, HandleState, OpenBrowse, OpenInputType, OpenInquire, OpenOptions, OpenSet, TaskNumber, TransactionId, UOWType, UserIdentifier

Response data for MQIACF_Q_STATUS:

CurrentQDepth (MQCFIN)
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

LastGetDate (MQCFST)
Date on which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_DATE).

The date, in the form yyyy-mm-dd, on which the last message was successfully read from the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastGetTime (MQCFST)
Time at which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_TIME).

The time, in the form hh.mm.ss, at which the last message was successfully read from the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

LastPutDate (MQCFST)
Date on which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_DATE).

The date, in the form yyyy-mm-dd, on which the last message was successfully put to the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastPutTime (MQCFST)
Time at which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_TIME).

The time, in the form hh.mm.ss, at which the last message was successfully put to the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

OldestMsgAge (MQCFIN)
Age of the oldest message (parameter identifier: MQIACF_OLDEST_MSG_AGE).

Age, in seconds, of the oldest message on the queue.

If the value is unavailable, MQMON_NOT_AVAILABLE is returned. If the queue is empty, 0 is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OnQTime (MQCFIL)
Indicator of the time that messages remain on the queue (parameter identifier: MQIACH_Q_TIME_INDICATOR). Amount of time, in microseconds, that a message spent on the queue. Two values are returned:

- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OpenInputCount (MQCFIN)
Open input count (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount (MQCFIN)
Open output count (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QueueMonitoring (MQCFIN)
Current level of monitoring data collection for the queue (parameter identifier: MQIA_MONITORING_Q). The value can be:
MQMON_OFF
Monitoring for the queue is switched off.
MQMON_LOW
Low rate of data collection.
MQMON_MEDIUM
Medium rate of data collection.
MQMON_HIGH
High rate of data collection.

StatusType (MQCFIN)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE). Specifies the type of status information.

UncommittedMsgs (MQCFIN)
Whether there are uncommitted messages (parameter identifier: MQIACF_UNCOMMITTED_MSGS). The value can be:
MQQSUM_YES
There are uncommitted messages.
MQQSUM_NO
There are no uncommitted messages.

Response data for MQIACF_Q_HANDLE:

ApplTag (MQCFST)
Open application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length of the string is MQ_APPL_TAG_LENGTH.

ApplType (MQCFIN)
Open application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:
MQAT_BATCH
Application using a batch connection.
MQAT_CICS
A CICS transaction.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Conname (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

HandleState (MQCFIN)
State of the handle (parameter identifier: MQIACF_HANDLE_STATE). The value may be:

MQHSTATE_ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.

MQHSTATE_INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

OpenBrowse (MQCFIN)
Open browse (parameter identifier: MQIACF_OPEN_BROWSE). The value can be:
MQQSO_YES
The queue is open for browsing.
MQQSO_NO
The queue is not open for browsing.

OpenInputType (MQCFIN)
Open input type (parameter identifier: MQIACF_OPEN_INPUT_TYPE). The value can be:
MQQSO_NO
The queue is not open for inputing.
MQQSO_SHARED
The queue is open for shared input.
MQQSO_EXCLUSIVE
The queue is open for exclusive input.

OpenInquire (MQCFIN)
Open inquire (parameter identifier: MQIACF_OPEN_INQUIRE). The value can be:
MQQSO_YES
The queue is open for inquiring.
MQQSO_NO
The queue is not open for inquiring.

OpenOptions (MQCFIN)
Open options currently in force for the queue (parameter identifier: MQIACF_OPEN_OPTIONS).

OpenOutput (MQCFIN)
Open output (parameter identifier: MQIACF_OPEN_OUTPUT). The value can be:
MQQSO_YES
The queue is open for outputting.
MQQSO_NO
The queue is not open for outputting.

OpenSet (MQCFIN)
Open set (parameter identifier: MQIACF_OPEN_SET). The value can be:
MQQSO_YES
The queue is open for setting.
MQQSO_NO
The queue is not open for setting.

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

StatusType (MQCFIN)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE). Specifies the type of status information.

TaskNumber (MQCFST)
CICS task number (parameter identifier: MQCACF_TASK_NUMBER). A 7-digit CICS task number as a 7-digit string with leading zero.

The length of the string is MQ_TASK_NUMBER_LENGTH.

TransactionId (MQCFST)
CICS transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID). A 4character CICS transaction identifier.

The length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWType (MQCFIN)
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be:

- MQUOWT_CICS

UserIdentifier (MQCFST)
Open application username (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Service (Response)

The response to the Inquire Service (MQCMD_INQUIRE_SERVICE) command consists of the response header followed by the ServiceName structure and the requested combination of attribute parameter structures. If a generic service name was specified, one such message is generated for each service found.

Always returned:
ServiceName

Returned if requested:
AlterationDate, AlterationTime, ServiceDesc, ServiceType, StartArguments, StartCommand, StartMode, StopArguments, StopCommand

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date on which the information was last altered in the form *yyyy-mm-dd*.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME). The time at which the information was last altered in the form *hh.mm.ss*.

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName (MQCFST)
Name of service definition (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ServiceType (MQCFIN)
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE). The value can be:
MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.
MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments (MQCFST)
Arguments to be passed in CICS COMMAREA when the CICS transaction is started (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is 100.

StartCommand (MQCFST)
CICS transaction to start the service (parameter identifier: MQCA_SERVICE_START_COMMAND).

The CICS transaction to start the service.

The maximum length of the string is 4.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be:
MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.
MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.
MQSVC_CONTROL_Q_MGR_START
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.

StopArguments (MQCFST)
The arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is 100.

StopCommand (MQCFST)
CICS transaction to stop the service (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

The maximum length of the string is 4.

Inquire Service Status (Response)

The response to the Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command consists of the response header followed by the ServiceName structure and the requested combination of attribute parameter structures. If a generic service name was specified, one such message is generated for each RUNNING service found.

Always returned:
ServiceName

Returned if requested:

ProcessId, ServiceDesc, StartArguments, StartCommand, StartDate, StartMode, StartTime, Status, StopArguments, StopCommand

Response data

ProcessId (MQCFIN)
Process identifier (parameter identifier: MQIACF_PROCESS_ID). The CICS task number associated with the service.

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCACH_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName (MQCFST)
Name of the service definition (parameter identifier: MQCA_SERVICE_NAME). The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

StartArguments (MQCFST)
Arguments to be passed to the program on startup (parameter identifier: MQCA_SERVICE_START_ARGS). The maximum length of the string is 100.

StartCommand (MQCFST)
CICS transaction code used to start service (parameter identifier: MQCA_SERVICE_START_COMMAND).

The maximum length of the string is 4.

StartDate (MQCFST)
Start date (parameter identifier: MQCACF_SERVICE_START_DATE). The date, in the form *yyyy-mm-dd*, on which the service was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode (MQCFIN) Service mode (parameter identifier: MQIACH_SERVICE_CONTROL). How the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.

MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

StartTime (MQCFST)
Start date (parameter identifier: MQCACF_SERVICE_START_TIME).

The time, in the form *hh.mm.ss*, at which the service was started.

The maximum length of the string is MQ_TIME_LENGTH.

Status (MQCFIN)
Service status (parameter identifier: MQIACH_SERVICE_STATUS). The current status of the service provided. The value can be:

MQSVC_STATUS_RUNNING
The service is running.

StopArguments (MQCFST)
Specifies the arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand (MQCFST)
CICS transaction code to stop service (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

The maximum length of the string is 4.

Inquire Subscription (Response)

The response to the Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command consists of the response header followed by the SubId and SubName structures, and the requested combination of attribute parameter structures (where applicable).

Always returned:

SubID, SubName

Returned if requested:

Destination, DestinationClass, DestinationCorrelId, DestinationQueueManager, PublishedAccountingToken, Expiry, PublishedApplicationIdentifier, PublishPriority, PublishSuscribeProperties, SubscriptionScope, TopicObject, TopicString, Userdata, VariableUser

Response data

AlterationDate (MQCFST)
The date of the most recent MQSUB or Change Subscription command that modified the properties of the subscription: yyyy-mm-dd.

AlterationTime (MQCFST)
The time of the most recent MQSUB or Change Subscription command that modified the properties of the subscription: hh:mm:ss.

CreationDate (MQCFST)
The creation date of the subscription, in the form yyyy-mm-dd.

CreationTime (MQCFST)
The creation time of the subscription, in the form hh.mm.ss.

Destination (MQCFST)
Destination (parameter identifier: MQCACF_DESTINATION). Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass (MQCFIN)
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS). Whether the destination is managed. The value can be:
MQDC_MANAGED
The destination is managed.
MQDC_PROVIDED
The destination queue is as specified in the Destination field.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID). A correlation identifier that is placed in the CorrelId field of the message descriptor for all the messages sent to this subscription. The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR). Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Durable (MQCFIN)
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

The value can be:

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY). A value of unlimited means that the subscription never expires. After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the AccountingToken field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN). The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationidentifier (MQCFST)
Value of the application identity data used in the ApplIdentityData field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA. The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of messages sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY). The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
The priority of messages sent to this subscription is taken from that supplied to the published message. This is the supplied default value.

PublishSubscribeProperties (MQCFIN)
Specifies how publish/subscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES). The value can be:
MQPSPROP_NONE
Publish/suscribe properties are not added to the messages. This is the supplied default value.
MQPSPROP_MSGPROP
Publish/suscribe properties are added as PCF attributes.

Reqonly(MQCFIN)
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY).
MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.
MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API

SubscriptionScope (MQCFST)
The scope of a subscription.

z/VSE always returns MQTSCOPE_QMGR.

SubscriptionType(MQCFIN)
Indicates how the subscription was created.

The value can be:
MQSUBTYPE_ADMIN
Created using DEF SUB MQSC or PCF command. This SUBTYPE also indicates that a subscription has been modified using an administrative command.
MQSUBTYPE_API
Created using an MQSUB API request.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription (parameter identifier: MQCACF_SUB_USER_ID). The maximum length of the string is MQ_USER_ID_LENGTH.

TopicObject (MQCFST)
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCACF_TOPIC). The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH 256.

VariableUser (MQCFIN)
Specifies whether a user other than the one who created the subscription, that is, the user shown in SubscriptionUser can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:
MQVU_ANY_USER
Any user can take over the ownership. This is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

Inquire Subscription Status (Response)

The response to the Inquire Subscription Status (MQCMD_INQUIRE_SBSTATUS) command consists of the response header followed by the SubId and SubName structures, and the requested combination of attribute parameter structures (where applicable).

Always returned:

None

Returned if requested:

ActiveConnection, Durable, LastPublishDate, LastPublishTime, NumberMsgs, ResumeDate, ResumeTime, SubID, SubType

Response data

ActiveConnection (MQCFST)
Returns the ConnId of the HConn that currently has this subscription open.

Durable (MQCFIN)
A durable subscription is not deleted when the creating application closes its subscription handle.

LastPublishDate (MQCFST)
The date on which a message was last published to the destination specified by this subscription.

LastPublishTime (MQCFST)
The time on which a message was last published to the destination specified by this subscription.

NumberMsgs (MQCFIN)
The number of messages put to the destination specified by this subscription.

ResumeDate (MQCFST)
The date of the most recent MQSUB API call that connected to the subscription.

ResumeTime (MQCFST)
The time of the most recent MQSUB API call that connected to the subscription.

SubID (MQCFBS)
The internal, unique key identifying a subscription.

SubType (MQCFIN)
Indicates how the subscription was created (parameter identifier: MQIA_SUB_TYPE).

MQSUBTYPE_ADMIN
Created using the DEF SUB MQSC or Create SubscriptionPCF command. This Subtype also indicates that a subscription has been modified using an administrative command.

MQSUBTYPE_API
Created using an MQSUB API call.

Inquire Topic (Response)

The response to the Inquire Topic (MQCMD_INQUIRE_TOPIC) command consists of the response header followed by the TopicName structure (and on z/OS only, the QSG Disposition structure), and the requested combination of attribute parameter structures (where applicable).

Always returned:

TopicName, TopicType

Returned if requested:

AlterationDate, AlterationTime, DefPersistence, DefPriority, DefPutResponse, DurableModelQName, DurableSubscriptions, InhibitPublications, InhibitSubscriptions, NonDurableModelQName, NonPersistentMsgDelivery, PersistentMsgDelivery

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE). The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY). Always returns 0 on z/VSE.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable managed subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB). The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE
Durable subscriptions are permitted.

MQSUB_NON_DURABLE
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB). The value can be:
MQTA_PUB_INHIBITED
Publications are inhibited for this topic.
MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB). The value can be:
MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.
MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable managed subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q). The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY). The value can be:
MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.
MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.
MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.
MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PublicationScope (MQCFIN)
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE). The value can be:
MQSCOPE_AS_PARENT
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic. MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.
MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

You can override this behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message Options.

Note: In z/VSE, publication can only be made to the local queue manager.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH

SubscriptionScope (MQCFIN)
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE). The value can be:

MQSCOPE_AS_PARENT
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic. MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC).

The maximum length is MQ_TOPIC_DESC_LENGTH.

TopicName (MQCFST)
Topic object name (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The "/" character within this string has special meaning. It delimits the elements in the topic tree. A topic string can start with the "/" character but is not required to. A string starting with the "/" character is not the same as the string which starts without the "/" character. A topic string cannot end with the "/" character.

The maximum length of the string is 256.

TopicType (MQCFIN)
Whether this object is a local or cluster topic (parameter identifier: MQIA_TOPIC_TYPE). The value can be:

MQTOPT_LOCAL
This object is a local topic.

Inquire Topic Names (Response)

The response to the Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified administrative topic name.

Always returned:

TopicNames

Returned if requested:

None

Response data

TopicNames (MQCFSL)

List of topic object names (parameter identifier: MQCACF_TOPIC_NAMES).

Inquire Topic Status (Response)

The response of the Inquire topic (MQCMD_INQUIRE_TOPIC_STATUS) command consists of the response header, followed by the TopicString structure, and the requested combination of attribute parameter structures (where applicable).

Always returned:

TopicString

Returned if requested and StatusType is MQIACF_TOPIC_STATUS:

DefPriority, DefaultPutResponse, DefPersistence, DurableSubscriptions, InhibitPublications, InhibitSubscriptions, AdminTopicName, DurableModelQName, NonDurableModelQName, PersistentMessageDelivery,RetainedPublication, PublishCount, SubscriptionScope, SubscriptionCount, PublicationScope

Returned if requested and StatusType is MQIACF_TOPIC_SUB:

SubscriptionId, SubscriptionUserId, Durable, SubscriptionType, ResumeDate, ResumeTime, LastMessageDate, LastMessageTime, NumberOfMessages, ActiveConnection

Returned if requested and StatusType is MQIACF_TOPIC_PUB:

LastPublishDate, LastPublishTime, NumberOfPublishes, ActiveConnection

Response data (TOPIC_STATUS)

DefPersistence (MQCFIN)

Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

Returned value:

MQPER_PERSISTENT

Message is persistent.

z/VSE messages are always persistent.

DefPriority (MQCFIN)

Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Shows the resolved default priority of messages published to the topic.

DurableSubscriptions (MQCFIN)

Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

Returned value:

MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.
MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

Returned value:
MQTA_PUB_INHIBITED
Publications are inhibited for this topic.
MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

Returned value:
MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.
MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

AdminTopicName (MQCFST)
Topic object name (parameter identifier: MQCA_ADMIN_TOPIC_NAME).

If the topic is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

DurableModelQName (MQCFST)
The name of the model queue used for managed durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

Shows the resolved value of the name of the model queue to be used for durable subscriptions that request the queue manager to manage the destination of publications.

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonDurableModelQName (MQCFST)
The name of the model queue for managed non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMessageDelivery (MQCFIN)
Delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

Returned value:
MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability, for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.
MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable

subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

RetainedPublication (MQCFIN)
Whether there is a retained publication for this topic (parameter identifier: MQIACF_RETAINED_PUBLICATION).

Returned value:

MQQSO_YES
There is a retained publication for this topic.

MQQSO_NO
There is no retained publication for this topic.

PublishCount (MQCFIN)
Publish count (parameter identifier: MQIA_PUB_COUNT).

The number of applications currently publishing to the topic.

SubscriptionCount (MQCFIN)
Subscription count (parameter identifier: MQIA_SUB_COUNT).

The number of subscribers for this topic string, including durable subscribers who are not currently connected.

SubscriptionScope (MQCFIN)
Determines whether this queue manager propagates subscriptions for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate subscriptions for this topic to other queue managers.

PublicationScope (MQCFIN)
Determines whether this queue manager propagates publications for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate publications for this topic to other queue managers.

Response data (TOPIC_STATUS_SUB)

SubscriptionId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

The queue manager assigns SubscriptionId as an all time unique identifier for this subscription.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

SubscriptionUserId (MQCFST)
The user ID that owns this subscription (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

Durable (MQCFIN)
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

SubscriptionType (MQCFIN)
The type of subscription (parameter identifier: MQIACF_SUB_TYPE).

The value can be:

MQSUBTYPE_ADMIN

MQSUBTYPE_API

ResumeDate (MQCFST)
Date of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

ResumeTime (MQCFST)
Time of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

LastMessageDate (MQCFST)
Date on which an MQPUT call last sent a message to this subscription. The queue manager updates the date field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_DATE). The maximum length of the string is MQ_DATE_LENGTH. Note that an MQSUBRQ call updates this value.

LastMessageTime (MQCFST)
Time at which an MQPUT call last sent a message to this subscription. The queue manager updates the time field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_TIME). The maximum length of the string is MQ_TIME_LENGTH. Note that an MQSUBRQ call updates this value.

NumberOfMessages (MQCFIN)
Number of messages put to the destination specified by this subscription (parameter identifier: MQIACF_MESSAGE_COUNT). Note that an MQSUBRQ call updates this value.

ActiveConnection (MQCFBS)
The currently active ConnectionId (CONNID) that opened this subscription (parameter identifier: MQBACF_CONNECTION_ID).

The maximum length of the string is MQ_CONNECTION_ID_LENGTH.

Response data (TOPIC_STATUS_PUB)

LastPublicationDate (MQCFST)
Date on which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

LastPublicationTime(MQCFST)
Time at which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

NumberOfPublishes(MQCFIN)
Number of publishes made by this publisher (parameter identifier: MQIACF_PUBLISH_COUNT).

ActiveConnection (MQCFBS)
The currently active ConnectionId (CONNID) associated with the handle that has this topic open for publish (parameter identifier: MQBACF_CONNECTION_ID).

Accounting and statistics messages

Accounting and statistics messages are generated intermittently by queue managers to record information about the MQI operations performed by WebSphere MQ applications, or to record information about the activities occurring in a WebSphere MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications.

Statistics messages
Statistics messages are used to record information about the activities occurring in a WebSphere MQ system.

Accounting and statistics messages are delivered to two system queues. Users or user applications can retrieve messages from these system queues and use the recorded information for various tasks such as resource charging or system monitoring.

Accounting messages

Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications. Every accounting message contains one or more records. A record is a PCF structure that holds information about a single activity performed by an application.

An accounting message is a WebSphere MQ message, formatted in PCF. 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.

The information contained within accounting messages can be used for the following:
- Account for application resource use.
- Record application activity.
- 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 your queue manager network is running correctly.

Accounting messages types

There are two categories of accounting message, as follows:

MQI accounting messages
MQI accounting messages contain information relating to the number of MQI requests executed using a connection to a queue manager.

Queue accounting messages
Queue accounting messages contain information relating to the number of MQI requests executed using connections to a queue manager, with respect to specific queues.

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.

Controlling accounting messages

The collection of accounting information is controlled by a set of queue manager, and queue attributes.

Collecting MQI accounting information
The collection of MQI accounting information is controlled by the queue manager attribute, ACCTMQI. To change the value of this attribute you can use the MQSC command, ALTER QMGR, and specify the parameter, ACCTMQI. This 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)

Collecting queue accounting information

The collection of queue accounting information is controlled by the queue attribute, ACCTQ, and the queue manager attribute, ACCTQ. To change the value of the queue attribute, you can use the MQSC command, ALTER QLOCAL and specify the parameter ACCTQ.

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 enable accounting information collection for the queue, Q1, use the following MQSC command:

```
ALTER QLOCAL(Q1) ACCTQ(ON)
```

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.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)
```

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.

Controlling accounting information collection using MQCONNX

The collection of both MQI and queue accounting information can also be modified at the connection level by specifying the ConnectOpts parameter on the MQCONNX call. By altering the value of ConnectOpts, it is possible to override the effective value of the queue manager attributes ACCTMQI and ACCTQ.

ConnectOpts can have the following values:

MQCNO_ACCOUNTING_MQI_ENABLED
If the value of the queue manager attribute ACCTMQI is specified as OFF, then 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, then this attribute has no effect.

MQCNO_ACCOUNTING_MQI_DISABLED
If the value of the queue manager attribute ACCTMQI is specified as ON, then 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, then this attribute has no effect.

MQCNO_ACCOUNTING_Q_ENABLED
If the value of the queue manager attribute ACCTQ is specified as OFF, then 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, then 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, then this attribute has no effect.

These overrides are disabled by default. To enable them, set the queue manager attribute ACCTCONO to ENABLED. To enable accounting overrides per connection use the following MQSC command:

```
ALTER QMGR ACCTCONO(ENABLED)
```

Generating accounting messages

Accounting messages are generated upon the disconnection of the application from the queue manager.

Intermediate accounting messages are also 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 only generated when the application interacts with the queue manager, so applications that remain connected to the queue manager for long periods without executing MQI requests will 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)
```

WebSphere MQ for z/VSE allows a minimum accounting interval of 10 seconds. A value less than 10 is ignored and 10 seconds is used.

Format of accounting messages

Accounting messages are constructed as a set of PCF fields that consist of the following:

A 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 is comprised of 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.

The parameters contained in MQI accounting message data are described in "MQI accounting message data" below.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and between one and one hundred 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.

The parameters contained in queue accounting message data are described in “Queue accounting message data” on page 453.

Statistics messages

Statistics messages are used to record information about the activities occurring in a WebSphere MQ system. Statistics messages are WebSphere MQ messages, formatted in PCF, that are delivered to the system queue (SYSTEM.ADMIN.STATISTICS.QUEUE) at configured intervals.

The information contained within statistics messages can be used for the following:
- 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 your queue manager network is running correctly.

Statistics message types

The various types of statistics message follow:

MQI statistics messages
MQI statistics messages contain information relating to the number of MQI requests executed during a configured interval. For example, the information can be the number of MQI commands executed 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.

Channel statistics messages
Channel statistics messages contain information relating to the activity of a channel during a configured interval. For example the information could 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 was collected.

Controlling statistics messaging: The collection of statistics data is controlled by queue manager, queue, and channel attributes.

Collecting MQI statistics information

The collection of MQI statistics information is controlled by the queue manager attribute, STATMQI. To change the value of this queue manager attribute, you can use the MQSC command, ALTER QMGR and specify the parameter STATMQI. STATMQI 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, use the following MQSC command:

```
ALTER QMGR STATMQI(ON)
```

Collecting queue statistics information

Queue statistics information collection can be enabled or disabled for individual queues, or for multiple queues. To control individual queues, the queue attribute STATQ must be set to enable or disable queue statistic information collection. To control many queues together, queue statistics information collection can be enabled or disabled at the queue manager level 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.

To change the value of the queue attribute STATQ, you can use the MQSC command, ALTER QLOCAL and specify the parameter STATQ. To change the value of the queue manager attribute STATQ, you can use the MQSC command, ALTER QMGR 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.

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.

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)
```

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.

Collecting channel statistics information

Channel statistics information collection can be enabled or disabled for individual channels, or for multiple channels. To control individual channels, the channel attribute STATCHL must be set to enable or disable channel statistic information collection. To control many channels together, channel statistics information collection can be enabled or disabled at the queue manager level 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.

Channel statistics information collection can be set to one of the three monitoring levels, low, medium or high. This is set at either object level, or at the queue manager level. The choice of which level to use is dependant on your system. Collecting statistics information data may require the execution of some relatively expensive instructions, so in order 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. The following table summarizes the levels available with channel statistics information collection:

Level	Description	Usage
Low	Measure a small sample of the data at regular intervals.	For channel objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most channel objects.
High	Measure all data, at regular intervals.	For channels 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, you can use the MQSC command, ALTER CHANNEL and specify the parameter STATCHL.

To change the value of the queue manager attribute STATCHL, you can use the MQSC command, ALTER QMGR and specify the parameter STATCHL. 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. This is the default value.

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.

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.

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)
```

Generating statistics messages

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. STATINT 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 you can you 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)
```

WebSphere MQ for z/VSE allows a minimum statistics interval of 10 seconds. A value less than 10 is ignored and 10 seconds is used.

Format of statistics messages

Statistics messages are constructed as a set of PCF fields that consist of the following:

A 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 is comprised of 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. The parameters contained in MQI statistics message data are described in "MQI statistics message data" on page 463.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and between one and one hundred 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.

The parameters contained in queue statistics message data are described in "Queue statistics message data" on page 470.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and between one and one hundred 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.

The parameters contained in channel statistics message data are described in "Channel statistics message data" on page 476.

Accounting and statistics message reference

This section provides an overview of the accounting and statistics message format. It describes the information returned in accounting messages, and in statistics messages.

Accounting and statistics message format

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.

Accounting and statistics messages contain the following:

A 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 MQMD (message descriptor)

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.

Datatype

MQCHAR8.

Value

MQFMT_ADMIN

Admin message.

Description

Format name of message data.

Datatype

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 parameters that make up the MQMD structure of event messages are described in the System Management Guide. 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 two parts:

- A PCF header (MQCFH).
- 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.

Datatype

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.
Datatype
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.

The parameters contained in MQI accounting message data are described in "MQI accounting message data" below.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and between one and one hundred QAccountingData PCF groups.

The parameters contained in queue accounting message data are described in "Queue accounting message data" below.

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

The parameters contained in MQI statistics message data are described in "MQI statistics message data" below.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters and between one and one hundred QStatisticsData PCF groups.

The parameters contained in queue statistics message data are described in "Queue statistics message data" below.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and between one and one hundred ChlStatisticsData PCF groups.

The parameters contained in channel statistics message data are described below.

MQI accounting message data

Message name
MQI accounting message.
System queue
SYSTEM.ADMIN.ACCOUNTING.QUEUE.

Accounting message data

QueueManager

Description
The name of the queue manager.

Identifier
MQCA_Q_MGR_NAME.

Datatype
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.

Datatype
MQCFST.

Maximum length
MQ_DATE_LENGTH.

Returned
Always.

IntervalStartTime

Description
The time of the start of the monitoring period.

Identifier
MQCAMO_START_TIME.

Datatype
MQCFST.

Maximum length
MQ_TIME_LENGTH.

Returned
Always.

IntervalEndDate

Description
The date of the end of the monitoring period.

Identifier
MQCAMO_END_DATE.

Datatype
MQCFST.

Maximum length
MQ_DATE_LENGTH.

Returned
Always.

IntervalEndTime

Description
The time of the end of the monitoring period.

Identifier
MQCAMO_END_TIME.

Datatype
MQCFST.

Maximum length
MQ_TIME_LENGTH.

Returned
Always.

CommandLevel
Description
The queue manager command level.
Identifier
MQIA_COMMAND_LEVEL.
Datatype
MQCFIN.
Returned
Always.

ConnectionId
Description
The connection identifier for the WebSphere MQ connection.
Identifier
MQBACF_CONNECTION_ID.
Datatype
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.
Datatype
MQCFIN.
Returned
Always.

ApplicationName
Description
The name of the application.
Identifier
MQCACF_APPL_NAME.
Datatype
MQCFST.
Maximum length
MQ_APPL_NAME_LENGTH.
Returned
Always.

ApplicationPid
Description
The operating system process identifier of the application.
Identifier
MQIACF_PROCESS_ID.
Datatype
MQCFIN.
Returned
Always.

ApplicationTid

Description
The WebSphere MQ thread identifier of the connection in the application.

Identifier
MQIACF_THREAD_ID.

Datatype
MQCFIN.

Returned
Always.

UserId

Description
The user identifier context of the application.

Identifier
MQCACF_USER_IDENTIFIER.

Datatype
MQCFST.

Maximum length
MQ_USER_ID_LENGTH.

Returned
Always.

ConnDate

Description
Date of MQCONN operation.

Identifier
MQCAMO_CONN_DATE.

Datatype
MQCFST.

Maximum length
MQ_DATE_LENGTH

Returned
When available.

ConnTime

Description
Time of MQCONN operation.

Identifier
MQCAMO_CONN_TIME.

Datatype
MQCFST.

Maximum length
MQ_TIME_LENGTH.

Returned
When available.

ConnName

Description
Connection name for client connection.

Identifier
MQCAMO_CONNECTION_NAME.

Datatype
MQCFST.

Maximum length
MQ_CONN_NAME_LENGTH.

Returned
When available.

ChannelName
Description
Channel name for client connection.
Identifier
MQCACH_CHANNEL_NAME.
Datatype
MQCFST.
Maximum length
MQ_CHANNEL_NAME_LENGTH.
Returned
When available.

DiscDate
Description
Date of MQDISC operation.
Identifier
MQCAMO_DISC_DATE.
Datatype
MQCFST.
Maximum length
MQ_DATE_LENGTH.
Returned
When available.

DiscTime
Description
Time of MQDISC operation.
Identifier
MQCAMO_DISC_TIME.
Datatype
MQCFST.
Maximum length
MQ_TIME_LENGTH.
Returned
When available.

DiscType
Description
Type of disconnect.
Identifier
MQIAMO_DISC_TYPE.
Datatype
MQCFIN.
Values
The possible values are:
MQDISCONNECT_TYPE_NORMAL
Requested by application.
MQDISCONNECT_TYPE_IMPLICIT
Abnormal application termination.
MQDISCONNECT_TYPE_QMGR
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 Note 1.

Identifier
MQIAMO_OPENS.
Datatype
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 Note 1.
Identifier
MQIAMO_OPENS_FAILED.
Datatype
MQCFIL.
Returned
When available.

CloseCount
Description
The number of objects closed. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_CLOSES.
Datatype
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 Note 1.
Identifier
MQIAMO_CLOSES_FAILED.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUTS.
Datatype
MQCFIL.
Returned
When available.

PutFailCount
Description
The number of unsuccessful attempts to put a message.

Identifier
MQIAMO_PUTS_FAILED.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUT1S.
Datatype
MQCFIL.
Returned
When available.

Put1FailCount
Description
The number of unsuccessful attempts to put a message using MQPUT1 calls.
Identifier
MQIAMO_PUT1S_FAILED.
Datatype
MQCFIN.
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 persistent value, see Note 2.
Identifier
MQIAMO64_PUT_BYTES.
Datatype
MQCFIL64.
Returned
When available.

GetCount
Description
The number of successful destructive gets for persistent and nonpersistent messages. This parameter is an integer list indexed by persistent value, see Note 2.
Identifier
MQIAMO_GETS.
Datatype
MQCFIL.
Returned
When available.

GetFailCount
Description
The number of unsuccessful destructive gets.

Identifier
MQIAMO_GETS_FAILED.
Datatype
MQCFIN.
Returned
When available.

GetBytes
Description
Total number of bytes retrieved for persistent and nonpersistent messages. This parameter is an integer list indexed by persistent value, see Note 2.
Identifier
MQIAMO64_GET_BYTES.
Datatype
MQCFIL64.
Returned
When available.

BrowseCount
Description
The number of successful non-destructive gets for persistent and nonpersistent messages. This parameter is an integer list indexed by persistent value, see Note 2.
Identifier
MQIAMO_BROWSES.
Datatype
MQCFIL.
Returned
When available.

BrowseFailCount
Description
The number of unsuccessful non-destructive gets.
Identifier
MQIAMO_BROWSES_FAILED.
Datatype
MQCFIN.
Returned
When available.

BrowseBytes
Description
Total number of bytes browsed for persistent and nonpersistent messages. This parameter is an integer list indexed by persistent value, see Note 2.
Identifier
MQIAMO64_BROWSE_BYTES.
Datatype
MQCFIL64.
Returned
When available.

CommitCount
Description
The number of successful calls to the MQCMIT MQI.
Identifier
MQIAMO_COMMITS.

Datatype
MQCFIN.
Returned
When available.

CommitFailCount
Description
The number of unsuccessful calls to the MQCMIT MQI.
Identifier
MQIAMO_COMMITS_FAILED.
Datatype
MQCFIN.
Returned
When available.

BackCount
Description
The number of calls to the MQBACK MQI.
Identifier
MQIAMO_BACKOUTS.
Datatype
MQCFIN.
Returned
When available.

InqCount
Description
The number of successful objects inquired upon. This attribute is an integer list which is indexed by the MQOT value for each object.
Identifier
MQIAMO_INQS.
Datatype
MQCFIL.
Returned
When available.

InqFailCount
Description
The number of unsuccessful object inquire attempts. This attribute is an integer list which is indexed by the MQOT value for each object.
Identifier
MQIAMO_INQS_FAILED.
Datatype
MQCFIL.
Returned
When available.

SetCount
Description
The number of successful MQSET calls. This attribute is an integer list which is indexed by the MQOT value for each object.
Identifier
MQIAMO_SETS.
Datatype
MQCFIL.

Returned
When available.

SetFailCount
Description
The number of unsuccessful MQSET calls. This attribute is an integer list which is indexed by the MQOT value for each object.
Identifier
MQIAMO_SETS_FAILED.
Datatype
MQCFIL.
Returned
When available.

Queue accounting message data

Message name
Queue accounting message.
System queue
SYSTEM.ADMIN.ACCOUNTING.QUEUE.

Accounting message data

QueueManager
Description
The name of the queue manager.
Identifier
MQCA_Q_MGR_NAME.
Datatype
MQCFST.
Max. length
MQ_Q_MGR_NAME_LENGTH.
Returned
Always.

IntervalStartDate
Description
The date of the start of the monitoring period.
Identifier
MQCAMO_START_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH.
Returned
Always.

IntervalStartTime
Description
The time of the start of the monitoring period.
Identifier
MQCAMO_START_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH.
Returned
Always.

IntervalEndDate

Description
The date of the end of the monitoring period.

Identifier
MQCAMO_END_DATE.

Datatype
MQCFST.

Max. length
MQ_DATE_LENGTH.

Returned
Always.

IntervalEndTime

Description
The time of the end of the monitoring period.

Identifier
MQCAMO_END_TIME.

Datatype
MQCFST.

Max. length
MQ_TIME_LENGTH.

Returned
Always.

CommandLevel

Description
The queue manager command level.

Identifier
MQIA_COMMAND_LEVEL.

Datatype
MQCFIN.

Returned
Always.

ConnectionId

Description
The connection identifier for the WebSphere MQ connection.

Identifier
MQBACF_CONNECTION_ID.

Datatype
MQCFBS.

Max. 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.

Datatype
MQCFIN.

Returned
Always.

ApplicationName

Description
The name of the application.

Identifier
MQCACF_APPL_NAME.

Datatype
MQCFST.

Max. length
MQ_APPL_NAME_LENGTH.

Returned
Always.

ApplicationPid

Description
The operating system process identifier of the application.

Identifier
MQIACF_PROCESS_ID.

Datatype
MQCFIN.

Returned
Always.

ApplicationTid

Description
The WebSphere MQ thread identifier of the connection in the application.

Identifier
MQIACF_THREAD_ID.

Datatype
MQCFIN.

Returned
Always.

UserId

Description
The user identifier context of the application.

Identifier
MQCACF_USER_IDENTIFIER.

Datatype
MQCFST.

Max. 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.

Datatype
MQCFIN.

Returned
Always.

QAccountingData

Description
Grouped parameters specifying accounting details for a queue.

Identifier
MQGACF_Q_ACCOUNTING_DATA.

Datatype
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.

Parameters of the QAccountingData group

Qname

Description
The name of the queue.

Identifier
MQCA_Q_NAME.

Datatype
MQCFST.

Included in PCF group
QAccountingData.

Max. length
MQ_Q_NAME_LENGTH.

Returned
When available.

CreateDate

Description
The date the queue was created.

Identifier
MQCA_CREATION_DATE.

Datatype
MQCFST.

Included in PCF group
QAccountingData

Max. length
MQ_DATE_LENGTH

Returned
When available.

CreateTime

Description
The time the queue was created.

Identifier
MQCA_CREATION_TIME.

Datatype
MQCFST.

Included in PCF group
QAccountingData.
Max. length
MQ_TIME_LENGTH.
Returned
When available.

Qtype
Description
The type of the queue.
Identifier
MQIA_Q_TYPE.
Datatype
MQCFIN.
Included in PCF group
QAccountingData.
Value MQQT_LOCAL.
Returned
When available.

QDefinitionType
Description
The queue definition type.
Identifier
MQIA_DEFINITION_TYPE.
Datatype
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.
Datatype
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 will reflect the date the queue was originally opened.
Identifier
MQCAMO_OPEN_DATE.

Datatype
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 will reflect the time the queue was originally opened.
Identifier
MQCAMO_OPEN_TIME.
Datatype
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 will not be returned.
Identifier
MQCAMO_CLOSE_DATE.
Datatype
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 will not be returned.
Identifier
MQCAMO_CLOSE_TIME.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUTS.

Datatype
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.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUT1S.
Datatype
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.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO64_PUT_BYTES.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUT_MIN_BYTES.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_PUT_MAX_BYTES.
Datatype
MQCFIL.
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 persistent value, see Note 2.
Identifier
MQIAMO_GETS.
Datatype
MQCFIL.
Included in PCF group
QAccountingData.
Returned
When available.

GetFailCount
Description
The number of unsuccessful destructive gets.
Identifier
MQIAMO_GETS_FAILED.
Datatype
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 persistent value, see Note 2.

Identifier
MQIAMO64_GET_BYTES.

Datatype
MQCFIL64.

Included in PCF group
QAccountingData.

Returned
When available.

GetMinBytes

Description
The size of the smallest persistent and nonpersistent message retrieved from the queue. This parameter is an integer list indexed by persistent value, see Note 2.

Identifier
MQIAMO_GET_MIN_BYTES.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

GetMaxBytes

Description
The size of the largest persistent and nonpersistent message retrieved from the queue. This parameter is an integer list indexed by persistent value, see Note 2.

Identifier
MQIAMO_GET_MAX_BYTES.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

BrowseCount

Description
The number of successful non-destructive gets for persistent and nonpersistent messages. This parameter is an integer list indexed by persistent value, see Note 2.

Identifier
MQIAMO_BROWSES.

Datatype
MQCFIL.

Included in PCF group
QAccountingData,

Returned
When available.

BrowseFailCount

Description
The number of unsuccessful non-destructive gets.
Identifier
MQIAMO_BROWSES_FAILED.
Datatype
MQCFIN.
Included in PCF group
QAccountingData.
Returned
When available.

BrowseBytes
Description
The number of bytes read in non-destructive gets that returned persistent messages.
Identifier
MQIAMO64_BROWSE_BYTES.
Datatype
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 persistent value, see Note 2.
Identifier
MQIAMO_BROWSE_MIN_BYTES.
Datatype
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 persistent value, see Note 2.

Identifier
MQIAMO_BROWSE_MAX_BYTES.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

TimeOnQMin
Description
The shortest time a persistent and nonpersistent message

remained on the queue before being retrieved, in microseconds. This parameter is an integer list indexed by persistent value, see Note 2 below. Note: For the purpose of determining the TimeOnQMin value, WebSphere MQ for z/VSE ignores messages older than one year.

Identifier
MQIAMO_Q_TIME_MIN.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

TimeOnQAvg

Description
The average time a persistent and nonpersistent message remained on the queue before being retrieved, in microseconds. This parameter is an integer list indexed by persistent value, see Note 2. Note: For the purpose of determining the TimeOnQAvg value, WebSphere MQ for z/VSE ignores messages older than one year.

Identifier
MQIAMO_Q_TIME_AVG.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

TimeOnQMax

Description
The longest time a persistent and nonpersistent message remained on the queue before being retrieved, in microseconds. This parameter is an integer list indexed by persistent value, see Note 2. Note: For the purpose of determining the TimeOnQMax value, WebSphere MQ for z/VSE ignores messages older than one year.

Identifier
MQIAMO_Q_TIME_MAX.

Datatype
MQCFIL.

Included in PCF group
QAccountingData.

Returned
When available.

MQI statistics message data

Message name
MQI statistics message.

System queue
SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager

Description:
Name of the queue manager.
Identifier:
MQCA_Q_MGR_NAME.
Datatype:
MQCFST.
Max. length:
MQ_Q_MGR_NAME_LENGTH.
Returned:
Always.

IntervalStartDate
Description
The date at the start of the monitoring period.
Identifier
MQCAMO_START_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH
Returned
Always.

IntervalStartTime
Description
The time at the start of the monitoring period.
Identifier
MQCAMO_START_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH
Returned
Always.

IntervalEndDate
Description
The date at the end of the monitoring period.
Identifier
MQCAMO_END_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH
Returned
Always.

IntervalEndTime
Description
The time at the end of the monitoring period.
Identifier
MQCAMO_END_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH
Returned
Always.

CommandLevel
Description
The queue manager command level.
Identifier
MQIA_COMMAND_LEVEL.
Datatype
MQCFIN.
Returned
Always.

ConnCount
Description
The number of successful connections to the queue manager.
Identifier
MQIAMO_CONNS.
Datatype
MQCFIN.
Returned
When available.

ConnFailCount
Description
The number of unsuccessful connection attempts.
Identifier
MQIAMO_CONNS_FAILED.
Datatype
MQCFIN.
Returned
When available.

ConnsMax
Description
The maximum number of concurrent connections in the recording interval.
Identifier
MQIAMO_CONNS_MAX.
Datatype
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_QMGR
Identifier
MQIAMO_DISCS.
Datatype
MQCFIL.
Returned
When available.

OpenCount

Description
The number of objects successfully opened. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_OPENS.
Datatype
MQCFIL.
Returned
When available.

OpenFailCount
Description
The number of unsuccessful open object attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_OPENS_FAILED.
Datatype
MQCFIL.
Returned
When available.

CloseCount
Description
The number of objects successfully closed. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_CLOSES.
Datatype
MQCFIL.
Returned
When available.

CloseFailCount
Description
The number of unsuccessful close object attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_CLOSES_FAILED.
Datatype
MQCFIL.
Returned
When available.

InqCount
Description
The number of objects successfully inquired upon. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_INQS.
Datatype
MQCFIL.
Returned
When available.

InqFailCount

Description
The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_INQS_FAILED.
Datatype
MQCFIL.
Returned
When available.

SetCount
Description
The number of objects successfully updated (SET). This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_SETS.
Datatype
MQCFIL.
Returned
When available.

SetFailCount
Description
The number of unsuccessful SET attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier
MQIAMO_SETS_FAILED.
Datatype
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 Note 2.
Identifier
MQIAMO_PUTS.
Datatype
MQCFIL.
Returned
When available.

PutFailCount
Description
The number of unsuccessful put message attempts.
Identifier
MQIAMO_PUTS_FAILED.
Datatype
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 Note 2.
Identifier
MQIAMO_PUT1S.
Datatype
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 Note 2.
Identifier
MQIAMO_PUT1S_FAILED.
Datatype
MQCFIN.
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 Note 2.
Identifier
MQIAMO64_PUT_BYTES.
Datatype
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 Note 2.
Identifier
MQIAMO_GETS.
Datatype
MQCFIL.
Returned
When available.

GetFailCount
Description
The number of unsuccessful destructive get requests.
Identifier
MQIAMO_GETS_FAILED.
Datatype
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 Note 2.

Identifier
MQIAMO64_GET_BYTES.

Datatype
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 Note 2.

Identifier
MQIAMO_BROWSES.

Datatype
MQCFIL.

Returned
When available.

BrowseFailCount

Description
The number of unsuccessful non-destructive get requests.

Identifier
MQIAMO_BROWSES_FAILED.

Datatype
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 Note 2.

Identifier
MQIAMO64_BROWSE_BYTES.

Datatype
MQCFIL64.

Returned
When available.

CommitCount

Description
The number of successful calls to the MQCMIT MQI.

Identifier
MQIAMO_COMMITS.

Datatype
MQCFIN.

Returned
When available.

CommitFailCount

Description
The number of unsuccessful calls to the MQCMIT MQI.
Identifier
MQIAMO_COMMITS_FAILED.
Datatype
MQCFIN.
Returned
When available.

BackCount
Description
The number of calls to the MQBACK MQI.
Identifier
MQIAMO_BACKOUTS.
Datatype
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. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier
MQIAMO_MQGS_EXPIRED.
Datatype
MQCFIL.
Returned
When available.

Queue statistics message data

Message name
Queue statistics message.
System queue
SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager
Description
Name of the queue manager.
Identifier
MQCA_Q_MGR_NAME.
Datatype
MQCFST.
Max. length
MQ_Q_MGR_NAME_LENGTH.
Returned
Always.

IntervalStartDate
Description
The date at the start of the monitoring period.
Identifier
MQCAMO_START_DATE.
Datatype
MQCFST.

Max. length
MQ_DATE_LENGTH
Returned
Always.

IntervalStartTime
Description
The time at the start of the monitoring period.
Identifier
MQCAMO_START_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH
Returned
Always.

IntervalEndDate
Description
The date at the end of the monitoring period.
Identifier
MQCAMO_END_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH
Returned
Always.

IntervalEndTime
Description
The time at the end of the monitoring period.
Identifier
MQCAMO_END_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH
Returned
Always.

CommandLevel
Description
The queue manager command level.
Identifier
MQIA_COMMAND_LEVEL.
Datatype
MQCFIN.
Returned
Always.

Qname
Description
The name of the queue.
Identifier
MQCA_Q_NAME.
Datatype
MQCFST.

Max. length
MQ_Q_NAME_LENGTH.
Returned
Always.

CreateDate
Description
The date when the queue was created.
Identifier
MQCA_CREATION_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH.
Returned
Always.

CreateTime
Description
The time when the queue was created.
Identifier
MQCA_CREATION_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH.
Returned
Always.

ObjectCount
Description
The number of queues 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.
Datatype
MQCFIN.
Returned
Always.

QStatisticsData

Description
Grouped parameters specifying statistics details for a queue.

Identifier
MQGACF_Q_STATISTICS_DATA.

Datatype
MQCFGR.

Parameters in group
QMinDepth QMaxDepth AvgTimeOnQ PutCount PutFailCount Put1Count Put1FailCount PutBytes GetCount GetFailCount GetBytes BrowseCount BrowseFailCount BrowseBytes ExpiredMsgCount NonQueuedMsgCount

Returned
Always.

Parameters of the QStatisticsData group

QMinDepth
Description
The minimum queue depth during the monitoring period.
Identifier
MQIAMO_Q_MIN_DEPTH.
Datatype
MQCFIN.
Included in PCF group
QStatisticsData.
Returned
When available.

QMaxDepth
Description
The maximum queue depth during the monitoring period.
Identifier
MQIAMO_Q_MAX_DEPTH.
Datatype
MQCFIN.
Included in PCF group
QStatisticsData.
Returned
When available.

AvgTimeOnQ
Description
The average latency, in microseconds, of messages retrieved from the queue during the monitoring period. Note: For the purpose of determining the AvgTimeOnQ value, WebSphere MQ for z/VSE ignores messages older than one year.
Identifier
MQIAMO_AVG_Q_TIME.
Datatype
MQCFIN.
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 Note 2.
Identifier
MQIAMO_PUTS.
Datatype
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.
Datatype
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 Note 2.
Identifier
MQIAMO_PUT1S.
Datatype
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.
Datatype
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.
Datatype
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 Note 2.
Identifier
MQIAMO_GETS.
Datatype
MQCFIL.
Included in PCF group
QStatisticsData.
Returned
When available.

GetFailCount
Description
The number of unsuccessful destructive get requests.
Identifier
MQIAMO_GETS_FAILED.
Datatype
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 Note 2.
Identifier
MQIAMO64_GET_BYTES.
Datatype
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 Note 2.
Identifier
MQIAMO_BROWSES.
Datatype
MQCFIL.
Included in PCF group
QStatisticsData.
Returned
When available.

BrowseFailCount
Description
The number of unsuccessful non-destructive get requests.
Identifier
MQIAMO_BROWSES_FAILED.

Datatype
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 Note 2.
Identifier
MQIAMO64_BROWSE_BYTES.
Datatype
MQCFIL64.
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. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier
MQIAMO_MSGS_EXPIRED.
Datatype
MQCFIL.
Included in PCF group
QStatisticsData.
Returned
When available.

Channel statistics message data

Message name
Channel statistics message.
System queue
SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager
Description
The name of the queue manager.
Identifier
MQCA_Q_MGR_NAME.
Datatype
MQCFST.
Max. length
MQ_Q_MGR_NAME_LENGTH.
Returned
Always.

IntervalStartDate
Description
The date at the start of the monitoring period.

Identifier
MQCAMO_START_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH.
Returned
Always.

IntervalStartTime
Description
The time at the start of the monitoring period.
Identifier
MQCAMO_START_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH.
Returned
Always.

IntervalEndDate
Description
The date at the end of the monitoring period
Identifier
MQCAMO_END_DATE.
Datatype
MQCFST.
Max. length
MQ_DATE_LENGTH.
Returned
Always.

IntervalEndTime
Description
The time at the end of the monitoring period
Identifier
MQCAMO_END_TIME.
Datatype
MQCFST.
Max. length
MQ_TIME_LENGTH
Returned
Always.

CommandLevel
Description
The queue manager command level.
Identifier
MQIA_COMMAND_LEVEL.
Datatype
MQCFIN.
Returned
Always.

ObjectCount
Description
The number of Channel objectsaccessed 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 RemoteQmgrName ConnectionName MsgCount TotalBytes NetTimeMin NetTimeAvg NetTimeMax ExitTimeMin ExitTimeAvg ExitTimeMax FullBatchCount IncmplBatchCount AverageBatchSize

Returned
Always.

ChannelName

Description
The name of the channel.

Identifier
MQCACH_CHANNEL_NAME.

Datatype
MQCFST.

Max. length
MQ_CHANNEL_NAME_LENGTH.

Returned
Always.

ChannelType

Description
The channel type.

Identifier
MQIACH_CHANNEL_TYPE.

Datatype
MQCFIN.

Values
Possible values are:

MQCHT_SENDER
Sender channel.

MQCHT_SERVER
Server channel.

MQCHT_RECEIVER
Receiver channel.

MQCHT_REQUESTER
Requester channel.

Returned
Always.

RemoteQmgr
Description
The name of the remote queue manager.
Identifier
MQCACH_REMOTE_Q_MGR_NAME.
Datatype
MQCFST.
Max. length
MQ_Q_MGR_NAME_LENGTH
Included in PCF group
ChlStatisticsData.
Returned
When available.

ConnectionName
Description
Connection name of remote queue manager.
Identifier
MQCACH_CONNECTION_NAME.
Datatype
MQCFST.
Max. length
MQ_CONN_NAME_LENGTH
Included in PCF group
ChlStatisticsData.
Returned
When available.

MsgCount
Description
The number of persistent and nonpersistent messages sent or received. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier
MQIAMO_MSGS.
Datatype
MQCFIL.
Included in PCF group
ChlStatisticsData.
Returned
When available.

TotalBytes
Description
The number of bytes sent or received for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2 below.
Identifier
MQIAMO64_BYTES.
Datatype
MQCFIL64.
Included in PCF group
ChlStatisticsData.
Returned
When available.

NetTimeMin
Description
The shortest recorded channel round trip measured in the recording interval, in microseconds.
Identifier
MQIAMO_NET_TIME_MIN.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is MEDIUM or HIGH).

NetTimeAvg
Description
The average recorded channel round trip measured in the recording interval, in microseconds.
Identifier
MQIAMO_NET_TIME_AVG.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is MEDIUM or HIGH).

NetTimeMax
Description
The longest recorded channel round trip measured in the recording interval, in microseconds.
Identifier
MQIAMO_NET_TIME_MAX.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is MEDIUM or HIGH).

ExitTimeMin
Description
The shortest recorded time, in microseconds, spent executing a user exit in the recording interval.
Identifier
MQIAMO_EXIT_TIME_MIN.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is HIGH).

ExitTimeAvg
Description
The average recorded time, in microseconds, spent executing a user exit in the recording interval.
Identifier
MQIAMO_EXIT_TIME_AVG.

Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is HIGH).

ExitTimeMax
Description
The longest recorded time, in microseconds, spent executing a user exit in the recording interval.
Identifier
MQIAMO_EXIT_TIME_MAX.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available (STATCHL is HIGH).

FullBatchCount
Description
The number of batches processed by the channel, that were sent because the value of the channel attribute BATCHSZ was reached.
Identifier
MQIAMO_FULL_BATCHES.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
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.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.
Returned
When available.

AverageBatchSize
Description
The average batch size of batches processed by the channel.
Identifier
MQIAMO_AVG_BATCHE_SIZE.
Datatype
MQCFIN.
Included in PCF group
ChlStatisticsData.

Returned
When available.

Notes about the message data structures

Note 1

This parameter relates to WebSphere MQ objects. This parameter is an array of values (MQCFIL) indexed by the constants below.

Note: The index is relative to zero so if using COBOL you need to add 1 to the values below, for example, MQOT-Q is the second integer of the array, so a value of 2 should be used.

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_STORAGE_CLASS (4)	Contains the value relating to storage class objects.
MQOT_Q_MGR (5)	Contains the value relating to queue manager objects.

Note 2

This parameter relates to WebSphere MQ messages. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the constants below.

Note: The index is relative to zero so if using COBOL you need to add 1 to the values below, for example, MQPER-PERSISTENT is the second integer of the array, so a value of 2 should be used.

Constant	Value
MQPER_NOT_PERSISTENT (0)	Contains the value for nonpersistent messages.
MQPER_PERSISTENT (1)	Contains the value for persistent messages.

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 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 whose monitoring attribute is specified with the default value, QMGR, real-time monitoring is controlled at the queue manager level.

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 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 impact 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 10 summarizes the monitoring levels available for real-time monitoring of channels:

Table 10. 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.	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.

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)
```

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)
```

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)
```

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)
```

Displaying queue and channel monitoring data

To display real-time monitoring information for a queue or channel, use either the WebSphere MQ Explorer, PCF 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, below, demonstrate how you can display monitoring data.

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:

1. To display real-time monitoring information for a queue, use either the WebSphere MQ Explorer, PCF Inquire Queue Status, 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, PCF Inquire Channel Status or the MQSC command DISPLAY CHSTATUS, specifying the optional parameter MONITOR.

Examples

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)
STOPREQ(NO)
RQMNAME(QM2)
```

Structures used for commands and responses

Commands and responses consist of a PCF header (MQCFH) structure followed by zero or more parameter structures. Each of these is one of:

PCF byte string parameter (MQCFBS).

PCF integer filter parameter (MQCFIF).

PCF integer parameter (MQCFIN).

PCF string parameter (MQCFST).

PCF integer list parameter (MQCFIL).

PCF string filter parameter (MQCFSF).

PCF string list parameter (MQCFSL).

This section defines these parameter structures.

MQCFH - PCF header

The MQCFH structure describes the information that is present at the start of the message data of a command message, or a response to a command message. In either case, the message descriptor Format field is MQFMT_ADMIN.

Type (MQLONG)
Structure type. This indicates the content of the message. These are valid:

MQCFT_COMMAND
Message is a command.

MQCFT_RESPONSE
Message is a response to a command.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFH structure. The value must be:

MQCFH_STRUC_LENGTH
Length of command format header structure.

The initial value of this field is MQCFH_STRUC_LENGTH.

Version (MQLONG)
Structure version number. The value must be:

MQCFH_VERSION_1
Version number for command format header structure.

Command (MQLONG), MQCFH structure
Command identifier. For a command message, this identifies the function to be performed. For a response message, it identifies the command to which this is the reply. These are valid:

MQCMD_CHANGE_CHANNEL
Change channel.

MQCMD_CHANGE_Q
Change queue.

MQCMD_CHANGE_Q_MGR
Change queue manager.

MQCMD_COPY_CHANNEL
Copy channel.

MQCMD_COPY_Q
Copy queue.

MQCMD_CREATE_CHANNEL
Create channel.

MQCMD_CREATE_Q
Create queue.

MQCMD_DELETE_CHANNEL
Delete channel.

MQCMD_DELETE_Q
Delete queue.

MQCMD_ESCAPE
Escape.

MQCMD_INQUIRE_CHANNEL
Inquire channel.

MQCMD_INQUIRE_CHANNEL_NAMES
Inquire channel names.

MQCMD_INQUIRE_CHANNEL_STATUS
Inquire channel status.

MQCMD_INQUIRE_CONNECTION
Inquire connection.

MQCMD_INQUIRE_Q
Inquire queue.

MQCMD_INQUIRE_Q_MGR
Inquire queue manager.

MQCMD_INQUIRE_Q_NAMES
Inquire queue names.

MQCMD_INQUIRE_Q_STATUS
Inquire queue status.

MQCMD_PING_Q_MGR
Ping queue manager.

MQCMD_RESET_CHANNEL
Reset channel.

MQCMD_START_CHANNEL
Start channel.

MQCMD_START_CHANNEL_LISTENER
Start channel listener.

MQCMD_STOP_CHANNEL
Stop channel.

MQCMD_STOP_CONNECTION
Stop connection.

MsgSeqNumber (MQLONG)
Message sequence number. This is the sequence number of the message within a group of related messages. For a command, this field must have the value one (because a command is always contained within a single message). For a response, the field has the value one for the first (or only) response to a command, and increases by one for each successive response to that command.

The last (or only) message in a group has the MQCFC_LAST flag set in the Control field.

The initial value of this field is 1.

Control (MQLONG)
Control options. These are valid:

MQCFC_LAST
Last message in the group.

For a command, this value must always be set.

MQCFC_NOT_LAST
Not the last message in the group.

The initial value of this field is MQCFC_LAST.

CompCode (MQLONG)
Completion code. This field is meaningful only for a response; its value is not significant for a command. These are possible:

MQCC_OK
Command completed successfully.

MQCC_WARNING
Command completed with warning.

MQCC_FAILED
Command failed.

MQCC_UNKNOWN
Whether command succeeded is not known.

The initial value of this field is MQCC_OK.

Reason (MQLONG)
Reason code qualifying completion code. This field is meaningful only for a response; its value is not significant for a command.

The possible reason codes that could be returned in response to a command are listed at the end of each command format described in "Definitions of the PCFs" on page 242.

The initial value of this field is MQRC_NONE.

ParameterCount (MQLONG)
Count of parameter structures. This is the number of parameter structures (MQCFIL, MQCFIN, MQCFSL, and MQCFST) that follow the MQCFH structure. The value of this field is zero or greater.

The initial value of this field is 0.

C language declaration

The C language declaration for the MQCFH data structure is:

```
typedef struct tagMQCFH {
        MQLONG  Type;            /* Structure type */
        MQLONG  StrucLength;     /* Structure length */
        MQLONG  Version;         /* Structure version number */
        MQLONG  Command;         /* Command identifier */
        MQLONG  MsgSeqNumber;    /* Message sequence number */
        MQLONG  Control;         /* Control options */
        MQLONG  CompCode;        /* Completion code */
        MQLONG  Reason;          /* Reason code qualifying completion code */
        MQLONG  ParameterCount;  /* Count of parameter structures */
} MQCFH;
```

COBOL language declaration

The COBOL language declaration for the MQCFH data structure is:

```
**   MQCFH structure
10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
```

```
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFH data structure is:

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

MQCFIF - PCF integer filter parameter

The MQCFIF structure describes an integer filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFIF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

Fields for MQCFIF:

Type (MQLONG)
Structure type.

This indicates that the structure is a MQCFIF structure describing an integer filter parameter. The value must be:

MQCFT_INTEGER_FILTER
Structure defining an integer filter.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIF structure. The value must be:

MQCFIF_STRUC_LENGTH
Length of command format integer-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:
MQCA_*
MQCACF_*
MQCAMO_*
MQCACH_*

Operator (MQLONG)
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:
MQCFOP_GREATER
Greater than.
MQCFOP_LESS
Less than.
MQCFOP_EQUAL
Equal to.
MQCFOP_NOT_EQUAL
Not equal to.
MQCFOP_NOT_LESS
Greater than or equal to.
MQCFOP_NOT_GREATER
Less than or equal to.
MQCFOP_CONTAINS
Contains a specified value. Use this when filtering on lists of integers.
MQCFOP_EXCLUDES
Does not contain a specified value. Use this when filtering on lists of integers.

See the FilterValue description for details telling you which operators may be used in which circumstances.

FilterValue (MQLONG)
Filter value identifier.

This specifies the filter-value that must be satisfied.

Depending on the parameter, the value and the permitted operators can be:

- An explicit integer value, if the parameter takes a single integer value. You can only use these operators:
  MQCFOP_GREATER
  MQCFOP_LESS
  MQCFOP_EQUAL
  MQCFOP_NOT_EQUAL
  MQCFOP_NOT_GREATER
  MQCFOP_NOT_LESS
- An MQ constant, if the parameter takes a single value from a possible set of values (for example, the value MQCHT_SENDER on the ChannelType parameter). You can only use MQCFOP_EQUAL or MQCFOP_NOT_EQUAL.

- An explicit value or an MQ constant, as the case may be, if the parameter takes a list of values. You can use either MQCFOP_CONTAINS or MQCFOP_EXCLUDES.

  For example, if the value 6 is specified with the operator MQCFOP_CONTAINS, all items where one of the parameter values is 6 are listed.

  Let's take another example. If you need to filter on queues that are enabled for put operations in your Inquire Queue command, the parameter is MQIA_INHIBIT_PUT, and the filter-value is MQQA_PUT_ALLOWED. The filter value must be a valid value for the parameter being tested.

  For language structure declarations refer to parts in the installation sublibrary:
  C CMQC.H
  COBOL CMQCIFL.C or CMQCFIFV.C
  PL/I CMQCFP.P

MQCFIN - PCF integer parameter

The MQCFIN structure describes an integer parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

Type (MQLONG)
Structure type. This indicates that the structure is a MQCFIN structure describing an integer parameter. The value must be:

MQCFT_INTEGER
Structure defining an integer.

The initial value of this field is MQCFT_INTEGER.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFIN structure. The value must be:

MQCFIN_STRUC_LENGTH
Length of command format integer-parameter structure.

The initial value of this field is MQCFIN_STRUC_LENGTH.

Parameter (MQLONG)
Parameter identifier. This identifies the parameter whose value is contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see “MQCFH - PCF header” on page 486 for details.

The initial value of this field is 0.

Value (MQLONG)
Parameter value. This is the value of the parameter identified by the Parameter field.

The initial value of this field is 0.

C language declaration

The C language declaration for the MQCFIN data structure is:

```
typedef struct tagMQCFIN {
        MQLONG  Type;         /* Structure type */
        MQLONG  StrucLength;  /* Structure length */
        MQLONG  Parameter;    /* Parameter identifier */
        MQLONG  Value;        /* Parameter value */
} MQCFIN;
```

COBOL language declaration

The COBOL language declaration for the MQCFIN data structure is:

```
**   MQCFIN structure
10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFIN data structure is:

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

MQCFSF - PCF string filter parameter

The MQCFSF structure describes a string filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFSF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

Fields for MQCFSF:

Type (MQLONG)
Structure type.

This indicates that the structure is a MQCFSF structure describing a string filter parameter. The value must be:

MQCFT_STRING_FILTER
Structure defining a string filter.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFSF structure. The value must be: MQCFSF_STRUC_LENGTH

This is the length, in bytes, of the MQCFSF structure, including the string at the end of the structure (the FilterValue field). The length must be a multiple of 4, and must be sufficient to contain the string. Bytes between the end of the string and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the FilterValue field:

MQCFSF_STRUC_LENGTH_FIXED
Length of fixed part of command format filter string-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:
MQCA_*
MQCACF_*
MQCAMO_*
MQCACH_*

Operator (MQLONG)
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:
MQCFOP_GREATER
Greater than.
MQCFOP_LESS
Less than.
MQCFOP_EQUAL
Equal to.
MQCFOP_NOT_EQUAL
Not equal to.
MQCFOP_NOT_LESS
Greater than or equal to.
MQCFOP_NOT_GREATER
Less than or equal to.
MQCFOP_LIKE
Matches a generic string.
MQCFOP_NOT_LIKE
Does not match a generic string.
MQCFOP_CONTAINS
Contains a specified string. Use this when filtering on lists of strings.
MQCFOP_EXCLUDES
Does not contain a specified string. Use this when filtering on lists of strings.
MQCFOP_CONTAINS_GEN
Contains an item which matches a generic string. Use this when filtering on lists of strings.
MQCFOP_EXCLUDES_GEN
Does not contain any item which matches a generic string. Use this when filtering on lists of strings.

See the FilterValue description for details telling you which operators may be used in which circumstances.

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the FilterValue field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier. The string data is in the character set defined by the CodedCharSetId field in the MQ header structure that precedes the MQCFH structure, or by the CodedCharSetId field in the MQMD if the MQCFH structure is at the start of the message.

FilterValueLength (MQLONG)
Length of filter-value string.

This is the length, in bytes, of the data in the FilterValue field. This must be zero or greater, and does not need to be a multiple of 4.

FilterValue (MQCHARxFilterValueLength)
Filter value.

This specifies the filter-value that must be satisfied. Depending on the parameter, the value and the permitted operators can be:

- An explicit string value. You can only use these operators:
  MQCFOP_GREATER MQCFOP_LESS
  MQCFOP_EQUAL
  MQCFOP_NOT_EQUAL
  MQCFOP_NOT_GREATER
  MQCFOP_NOT_LESS
- A generic string value.

  This is a character string with an asterisk at the end, for example ABC*. The operator must be either MQCFOP_LIKE or MQCFOP_NOT_LIKE. The characters must be valid for the attribute you are testing. If the operator is MQCFOP_LIKE, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is MQCFOP_NOT_LIKE, all items where the attribute value does not begin with the string are listed. If the parameter takes a list of string values, the operator can be:
  MQCFOP_CONTAINS
  MQCFOP_EXCLUDES
  MQCFOP_CONTAINS_GEN
  MQCFOP_EXCLUDES_GEN
- An item in a list of values.

  The value can be explicit or or generic. If it is explicit, use MQCFOP_CONTAINS or MQCFOP_EXCLUDES as the operator. For example, if the value DEF is specified with the operator MQCFOP_CONTAINS, all items where one of the attribute values is DEF are listed. If it is generic, use MQCFOP_CONTAINS_GEN or MQCFOP_EXCLUDES_GEN as the operator. If ABC* is specified with the operator MQCFOP_CONTAINS_GEN, all items where one of the attribute values begins with ABC are listed.

Note:

1. If the specified string is shorter than the standard length of the parameter in MQFMT_ADMIN command messages, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
2. When the queue manager reads an MQCFSF structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes the string as though it had been specified on an MQI call. This means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.
3. The filter value must be a valid value for the parameter being tested.

MQCFST - PCF string parameter

The MQCFST structure describes a string parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The structure ends with a variable-length character string; see the String field below for further details.

Type (MQLONG)
Structure type. This indicates that the structure is an MQCFST structure describing a string parameter. The value must be:

MQCFT_STRING
Structure defining a string.

The initial value of this field is MQCFT_STRING.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFST structure, including the String field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the String field:

MQCFST_STRUC_LENGTH_FIXED
Length of fixed part of command format string-parameter structure.

The initial value of this field is MQCFST_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier. This identifies the parameter whose value is contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see "MQCFH - PCF header" on page 486 for details.

The initial value of this field is 0.

CodedCharSetId (MQLONG)
Coded character set identifier. This specifies the coded character set identifier of the data in the String field. This special value can be used:

MQCCSI_DEFAULT
Default coded character set identifier.

Character data is in the character set defined by the CodedCharSetId field in the message descriptor MQMD.

The initial value of this field is MQCCSI_DEFAULT.

StringLength (MQLONG)
Length of string. This is the length in bytes of the data in the String field; it must be zero or greater. This length need not be a multiple of four.

The initial value of this field is 0.

String (MQCHAR××StringLength)
String value. This is the value of the parameter identified by the Parameter field:

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, those characters in excess of the standard length must be blanks.
- In MQFMT_ADMIN response messages, string parameters are returned padded with blanks to the standard length of the parameter.

The string can contain any characters that are in the character set defined by CodedCharSetId, and that are valid for the parameter identified by Parameter.

Note: In the MQCFST structure, a null character in the string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_ADMIN message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

In contrast, when the queue manager reads an MQFMT_ADMIN message from the command input queue, the queue manager processes the data as though it had been specified on an MQI call. This means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL and PL/I programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFST in a larger structure, and declare additional field(s) following MQCFST, to represent the String field as required.

In C, the initial value of this field is the null string.

C language declaration

The C language declaration for the MQCFST data structure is:

```
typedef struct tagMQCFST {
        MQLONG  Type;           /* Structure type */
        MQLONG  StrucLength;    /* Structure length */
        MQLONG  Parameter;      /* Parameter identifier */
```

```
        MQLONG  CodedCharSetId;  /* Coded character set identifier */
        MQLONG  StringLength;    /* Length of string */
        MQCHAR  String[1];       /* String value - first character */
} MQCFST;
```

COBOL language declaration

The COBOL language declaration for the MQCFST data structure is:

```
**   MQCFST structure
10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFST data structure is:

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

MQCFIL - PCF integer list parameter

The MQCFIL structure describes an integer-list parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The structure ends with a variable-length array of integers; see the Values field below for further details.

Type (MQLONG)
Structure type. This indicates that the structure is an MQCFIL structure describing an integer-list parameter. The value must be:

MQCFT_INTEGER_LIST
Structure defining an integer list.

The initial value of this field is MQCFT_INTEGER_LIST.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFIL structure, including the Values field). The length must be a multiple of four, and must be sufficient to contain the array; any bytes between the end of the array and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the Values field:

MQCFIL_STRUC_LENGTH_FIXED
Length of fixed part of command format integer-list parameter structure.

The initial value of this field is MQCFIL_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier. This identifies the parameter whose values are contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see "MQCFH - PCF header" on page 486 for details.

The initial value of this field is 0.

Count (MQLONG)
Count of parameter values. This is the number of elements in the Values array; it must be zero or greater.

The initial value of this field is 0.

Values (MQLONG×Count)
Parameter values. This is an array of values for the parameter identified by the Parameter field. For example, for MQIACF_Q_ATTRS, this is a list of attribute selectors (MQCA_* and MQIA_* values).

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL and PL/I programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFIN in a larger structure, and declare additional field(s) following MQCFIN, to represent the Values field as required.

In C, the initial value of this field is a single 0.

C language declaration

The C language declaration for the MQCFIL data structure is:

```
typedef struct tagMQCFIL {
        MQLONG  Type;         /* Structure type */
        MQLONG  StrucLength;  /* Structure length */
        MQLONG  Parameter;    /* Parameter identifier */
        MQLONG  Count;        /* Count of parameter values */
        MQLONG  Values[1];    /* Parameter values - first element */
} MQCFIL;
```

COBOL language declaration

The COBOL language declaration for the MQCFIL data structure is:

```
**   MQCFIL structure
10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFIL data structure is:

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

MQCFSL - PCF string list parameter

The MQCFSL structure describes a string-list parameter in a message which is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The structure ends with a variable-length array of character strings; see the Strings field below for further details.

Type (MQLONG)
Structure type. This indicates that the structure is an MQCFSL structure describing a string-list parameter. The value must be:

MQCFT_STRING_LIST
Structure defining a string list.

The initial value of this field is MQCFT_STRING_LIST.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFSL structure, including the Strings field). The length must be a multiple of four, and must be sufficient to contain all of the strings; any bytes between the end of the strings and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the Strings field:

MQCFSL_STRUC_LENGTH_FIXED
Length of fixed part of command format string-list parameter structure.

The initial value of this field is MQCFSL_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier. This identifies the parameter whose values are contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see "MQCFH - PCF header" on page 486 for details.

The initial value of this field is 0.

CodedCharSetId (MQLONG)
Coded character set identifier. This specifies the coded character set identifier of the data in the Strings field. This special value can be used:

MQCCSI_DEFAULT
Default coded character set identifier. Character data is in the character set defined by the CodedCharSetId field in the message descriptor MQMD.

The initial value of this field is MQCCSI_DEFAULT.

Count (MQLONG)
Count of parameter values. This is the number of strings present in the Strings field; it must be zero or greater.

The initial value of this field is 0.

StringLength (MQLONG)
Length of one string. This is the length in bytes of one parameter value, that is the length of one string in the Strings field; all of the strings are this length. The length must be zero or greater, and need not be a multiple of four.

The initial value of this field is 0.

Strings (MQCHAR×StringLength×Count)
String values. This is a set of string values for the parameter identified by the Parameter field. The number of strings is given by the Count field, and the length of each string is given by the StringLength field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, StringLength×Count).

In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, those characters in excess of the standard length must be blanks.

In MQFMT_ADMIN response messages, string parameters are returned padded with blanks to the standard length of the parameter.

In all cases, StringLength gives the length of the string actually present in the message.

The strings can contain any characters that are in the character set defined by CodedCharSetId, and that are valid for the parameter identified by Parameter.

Note: In the MQCFSL structure, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_ADMIN message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

In contrast, when the queue manager reads an MQFMT_ADMIN message from the command input queue, the queue manager processes the data as though it had been specified on an MQI call. This means that within each string, the first null and the characters following it (up to the end of the string) are treated as blanks.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL and PL/I programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFSL in a larger structure, and declare additional field(s) following MQCFSL, to represent the Strings field as required.

In C, the initial value of this field is the null string.

C language declaration

The C language declaration for the MQCFSL data structure is:

```
typedef struct tagMQCFSL {
        MQLONG  Type;            /* Structure type */
        MQLONG  StrucLength;     /* Structure length */
        MQLONG  Parameter;       /* Parameter identifier */
        MQLONG  CodedCharSetId;  /* Coded character set identifier */
        MQLONG Count;           /* Count of parameter values */
        MQLONG  StringLength;    /* Length of one string */
        MQCHAR  Strings[1];      /* String values - first character */
} MQCFSL;
```

COBOL language declaration

The COBOL language declaration for the MQCFSL data structure is:

```
**   MQCFSL structure
10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFSL data structure is:

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

MQCFBS - PCF byte string parameter

The MQCFBS structure describes a byte-string parameter in a PCF message. The format name in the message descriptor is MQFMT_ADMIN.

When an MQCFBS structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_2 or greater.

In a user PCF message, the Parameter field has no significance, and can be used by the application for its own purposes.

The structure ends with a variable-length byte string.

Fields for MQCFBS

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFBS structure describing byte string parameter. The value must be:

MQCFT_BYTE_STRING Structure defining a byte string.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFBS structure, including the variable-length string at the end of the structure (the String field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the String field:

MQCFBS_STRUC_LENGTH_FIXED Length of fixed part of MQCFBS structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter whose value is contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see MQCFH - PCF header for details.

StringLength (MQLONG)
Length of string.

This is the length in bytes of the data in the string field; it must be zero or greater. This length need not be a multiple of four.

String (MQBYTExStringLength)
String value.

This is the value of the parameter identified by the parameter field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.

Note: A null character in the string is treated as normal data, and does not act as a delimiter for the string.

For MQFMT_ADMIN messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be nulls. If the specified string is longer than the standard length, it is an error.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For other programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFBS in a larger structure, and declare additional fields following MQCFBS, to represent the String field as required.

C language declaration

```
typedef struct tagMQCFBS {
        MQLONG  Type;         /* Structure type */
        MQLONG  StrucLength;  /* Structure length */
        MQLONG  Parameter;    /* Parameter identifier */
        MQLONG  StringLength; /* Length of string */
        MQBYTE  String[1];    /* String value - first byte */
} MQCFBS;
```

COBOL language declaration

```
**   MQCFBS structure
  10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER   PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFBS based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31)  /* Length of string */
```

MQCFIL64 - PCF 64-bit integer list parameter

The MQCFIL64 structure describes an 64-bit integer-list parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The structure ends with a variable-length array of 64-bit integers; see the Values field below for further details.

Type (MQLONG)
Structure type. This indicates that the structure is an MQCFIL64 structure describing a 64-bit integer-list parameter. The value must be:

MQCFT_INTEGER64_LIST
Structure defining a 64-bit integer list.

The initial value of this field is MQCFT_INTEGER64_LIST.

StrucLength (MQLONG)
Structure length. This is the length in bytes of the MQCFIL64 structure, including the Values field). The length must be a multiple of four, and must be sufficient to contain the array; any bytes between the end of the array and the length defined by the StrucLength field are not significant.

The following constant gives the length of the fixed part of the structure, that is the length excluding the Values field:

MQCFIL64_STRUC_LENGTH_FIXED
Length of fixed part of command format 64-bit integer-list parameter structure.

The initial value of this field is MQCFIL64_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier. This identifies the parameter whose values are contained in the structure. The values that can occur in this field depend on the value of the Command field in the MQCFH structure; see "MQCFH - PCF header" on page 486 for details.

The initial value of this field is 0.

Count (MQLONG)
Count of parameter values. This is the number of elements in the Values array; it must be zero or greater.

The initial value of this field is 0.

Values (MQLONGxCount)
Parameter values. This is an array of values for the parameter identified by the Parameter field.

The way that this field is declared depends on the programming language:
- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL and PL/I programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFIL64 in a larger structure, and declare additional field(s) following MQCFIL64, to represent the Values field as required.

In C, the initial value of this field is a single 0.

C language declaration

The C language declaration for the MQCFIL64 data structure is:

```
typedef struct tagMQCFIL64 {
        MQLONG  Type;        /* Structure type */
        MQLONG  StrucLength; /* Structure length */
        MQLONG  Parameter;   /* Parameter identifier */
        MQLONG  Count;       /* Count of parameter values */
        MQINT64 Values[1];   /* Parameter values - first element */
} MQCFIL64;
```

COBOL language declaration

The COBOL language declaration for the MQCFIL64 data structure is:

```
** MQCFIL structure
  10 MQCFIL64.
** Structure type
   15 MQCFIL64-TYPE PIC S9(9) BINARY.
** Structure length
   15 MQCFIL64-STRUCLENGTH PIC S9(9) BINARY.
** Parameter identifier
   15 MQCFIL64-PARAMETER PIC S9(9) BINARY.
** Count of parameter values
   15 MQCFIL64-COUNT PIC S9(9) BINARY.
```

PL/I language declaration

The PL/I language declaration for the MQCFIL64 data structure is:

```
dcl
 1 MQCFIL64 based,
  3 Type        fixed bin(31), /* Structure type */
```

```
3 StrucLength fixed bin(31), /* Structure length */
3 Parameter   fixed bin(31), /* Parameter identifier */
3 Count       fixed bin(31); /* Count of parameter values */
```

MQCFIL64 - PCF 64-bit integer list parameter

Chapter 9. WebSphere MQ commands

WebSphere MQ commands (MQSC) provide a uniform method of issuing human-readable commands on WebSphere MQ platforms.

This chapter describes:

Review section "Features" on page 15 for prerequisites for this feature.

Rules for using WebSphere MQ commands

You should observe the following rules when using WebSphere MQ 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 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. There must be at least one blank immediately preceding each parameter.
- Any number of blanks can occur at the beginning or end of the command, and between parameters, punctuation, and values. For example, this command is valid:

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

  Blanks within a pair of quotation marks are significant.
- Repeated parameters are not allowed.
- Strings that contain non-alphanumeric characters must be enclosed in single quotation marks.
- A string containing no characters (that is, two single quotation marks with no space in between) is not valid.
- A left parenthesis followed by a right parenthesis, with no significant information in between. For example `NAME ()` is not valid.
- Keywords are not case sensitive — AltER, alter, and ALTER are all acceptable. Names that are not contained within quotation marks are converted 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.

The following characters have special meaning when you build WebSphere MQ commands:

Table 11. MQSC special characters

Character	Description
blank	Blanks are used as separators. Multiple blanks are equivalent to a single blank, except in strings that have quotation marks (') round them.
'	A single quotation mark indicates the beginning or end of a string. WebSphere MQ leaves all characters that have quotation marks round them exactly as they are entered. The containing quotation marks are not included when calculating the length of the string.
''	Two quotation marks together inside a string are treated by WebSphere MQ as one quotation mark, and the string is not terminated. The double quotation marks are treated as one character when calculating the length of the string.
(	An open parenthesis indicates the beginning of a parameter list.
)	A close parenthesis indicates the end of a parameter list.
*	An asterisk indicates a wild card when used in an object name with DISPLAY commands for queues and channels. An asterisk is permitted in column 1 of input cards to indicate a comment.
+	A plus indicates continuation for commands that span more than one input card.

Issuing WebSphere MQ commands

With WebSphere MQ for z/VSE, WebSphere MQ commands are issued from a batch job, or via PCF Escape commands. WebSphere MQ for z/VSE provides a batch utility program (MQPMQSC) which can be used to issue WebSphere MQ commands from batch.

MQSC utility program

The MQSC utility program (MQPMQSC) uses the WebSphere MQ for z/VSE batch interface. Consequently, to issue WebSphere MQ commands using the MQPMQSC program, the batch interface, and the WebSphere MQ queue manager, must be active in CICS.

The MQPMQSC program reads WebSphere MQ commands from SYSIPT (all 80 columns are parsed), converts them into PCF Escape messages and puts them on the system command queue. Responses to WebSphere MQ commands issued this way are sent to the system reply queue. Both the system command queue and the system reply queue are specified as part of the queue manager's global system definition.

The system command and reply queues can be displayed and modified using PCF commands, or the MQMT transaction, option 1.1, followed by PF9:

```
2011/10/31        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
22:07:18                  Global System Definition                  CIC1
MQWMSYS                    Communications Settings                  A000

    TCP/IP settings                          Batch Interface settings
    Licensed clients . . : 00000             Batch Int. identifier: MQBSRV39
    Adopt MCA  . . . . . : N                 Batch Int. auto-start: Y
    Adopt MCA Check  . . : N
                                             Channel Auto-Definition
                                             Auto-definition  . . : N
    SSL parameters                           Auto-definition exit :
    Key-ring sublibrary  :
    Key-ring member  . . :
    SSL reset count  . . :

    PCF parameters
    System command queue : SYSTEM.ADMIN.COMMAND.QUEUE
    System reply queue . : SYSTEM.ADMIN.REPLY.QUEUE
    Cmd Server auto-start: Y
    Cmd Server convert . : Y
    Cmd Server DLQ store : Y

Requested record displayed.
PF2=Return PF3=Quit PF4/Enter=Read PF6=Update PF10=Listeners PF11=Services
```

Figure 82. System command and reply queues

Since WebSphere MQ commands are issued as PCF Escape messages, the PCF command server (transaction MQCS) must be active in CICS. The command server processes the PCF Escape messages as they arrive on the system command queue, and places PCF Escape responses on the system reply queue. If the command server is not active, the MQPMQSC command will timeout, and issue an appropriate error response.

The MQPMQSC program reports the results of the WebSphere MQ commands to SYSLST.

MQPMQSC sample JCL

The MQPMQSC program is provided with WebSphere MQ for z/VSE and resides in the installation library (default PRD2.WMQZVSE). The program should be run in a batch partition.

The following sample JCL illustrates how an WebSphere MQ command (ALTER QLOCAL) can be issued from a batch job:

```
// JOB MQSCJOB
// SETPARM MQBISRV='MQBISRV2'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// EXEC MQPMQSC,SIZE=AUTO
ALTER QLOCAL('ANYQ') GET(DISABLED)
/*
/&
```

In this sample, the SETPARM identifies the batch interface server by name. This should match the batch interface identifier specified in the appropriate queue manager's global system definition.

The MQPMQSC program can process multiple WebSphere MQ commands, for example:

```
// JOB MQSCJOB
// SETPARM MQBISRV='MQBISRV2'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// EXEC MQPMQSC,SIZE=AUTO
ALTER QLOCAL('ANYQ') GET(DISABLED)
DELETE CHANNEL('VSE1.TO.NT5')
DISPLAY QMGR CCSID
/*
/&
```

In addition, WebSphere MQ commands can be split over multiple SYSIPT lines by using the plus character (+) to indicate continuation. For example:

```
// JOB MQSCJOB
// SETPARM MQBISRV='MQBISRV2'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// EXEC MQPMQSC,SIZE=AUTO
ALTER CHANNEL('VSE1.TO.NT5') +
      CHLTYPE(SDR)           +
      CONVERT(NO)
/*
/&
```

WebSphere MQ command prerequisites

There are several prerequisites that must be met before WebSphere MQ commands can be processed by the MQPMQSC utility program. These include:

- WebSphere MQ is installed and active in CICS.
  See Chapter 2, “Installation,” on page 13.
- System command queue is defined to the queue manager.
  See “Queue Manager Communications Settings” on page 85.
- System reply queue is defined to the queue manager.
  See “Queue Manager Communications Settings” on page 85.
- Batch interface is active in CICS.
  See “Using the batch interface” on page 178.
- PCF command server is active in CICS.
  See “Preparing WebSphere MQ for PCF” on page 235.

Descriptions of the WebSphere MQ commands

This section describes the WebSphere MQ commands supported by WebSphere MQ for z/VSE.

WebSphere MQ commands can be divided into these categories:

- WebSphere MQ channel commands
- WebSphere MQ channel authentication commands
- WebSphere MQ channel listener commands
- WebSphere MQ connection commands
- WebSphere MQ namelist commands
- WebSphere MQ queue commands
- WebSphere MQ queue manager commands
- WebSphere MQ services commands
- WebSphere MQ subscription commands
- WebSphere MQ topic commands
- WebSphere MQ meta commands

Note: In the following command descriptions, where a parameter requires an integer value, but a character string is provided, the value is interpreted as zero.

A new keyword, WHERE, is provided for the MQSC DISPLAY commands. This allows you to filter the information displayed by one (and only one) of the attributes of objects.

The following describes the WHERE parameter that may be added to DISPLAY commands to filter the output.

WHERE

The filter condition is in three parts: filter-keyword, operator, and filter-value:

Filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command.

Operator
This is used to determine whether a channel satisfies the filter value on the given filter keyword. The operators are:

LT Less than.
GT Greater than.
EQ Equal to.
NE Not equal to.
LE Less than or equal to.
GE Greater than or equal to.
LK Matches a generic string that you provide as a filter-value.
NL Does not match a generic string that you provide as a filter-value.
CT Contains a specified item. If the filter-keyword is a list, you can use this to display objects the attributes of which contain the specified item.
EX Does not contain a specified item. If the filter-keyword is a list, you can use this to display objects the attributes of which do not contain the specified item.
CTG Contains an item which matches a generic string that you provide as a filter-value. If the filter-keyword is a list, you can use this to display objects the attributes of which match the generic string.
EXG Does not contain any item which matches a generic string that you provide as a filter-value. If the filter-keyword is a list, you can use this to display objects the attributes of which do not match the generic string.

Filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested. You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the TYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end; for example, ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

WebSphere MQ channel commands

The WebSphere MQ channel commands are:

ALTER CHANNEL
DEFINE CHANNEL
DELETE CHANNEL
DISPLAY CHANNEL
DISPLAY CHSTATUS
RESET CHANNEL
START CHANNEL
STOP CHANNEL

ALTER CHANNEL

Purpose

Use ALTER CHANNEL to alter the parameters of a channel.

Synonym

```
ALT CHL
```

Syntax

```
ALTER CHANNEL(channel-name) CHLTYPE(channel-type) optional-parameters
```

Parameters

channel-name
Channel name. The channel-name value should match the name of a channel defined to the queue manager.

channel-type
Channel type. The channel-type value should match the channel type of the channel identified by channel-name. Valid channel types include:

RCVR Receiver.

RQSTR
Requester.

SDR Sender.

SVR Server.

SVRCONN
Server connection (used by clients).

optional-parameters
Optional parameters for the ALTER CHANNEL command include:

BATCHINT(integer)
Batch interval.

BATCHSZ(integer)
Batch size.

DESCR(string)
Description.

CONNAME(string)
Connection name.

CONVERT(NO/YES)
Whether sender should convert application data.

DISCINT(integer)
Disconnection interval.

DISCRTY(integer)
Disconnection retry count.

LONGRTY(integer)
Long connection retry count.

LONGTMR(integer)
Long connection retry interval.

MAXMSGL(integer)
Maximum message length.

MAXXMIT(integer)
Maximum transmission size.

MONC(QMGR/OFF/LOW/MEDIUM/HIGH)
Channel monitoring setting.

MSGDATA (string)
Message exit user data.

MSGEXIT (string)
Message exit name.

PORTNUM(integer)
TCP/IP port number.

PROPCTL(COMPAT/ALL/NONE/)
Property control attribute.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender and Server channels and is optional. Permitted values are:

COMPAT
This is the default value.

Message properties	Result
The message contains a property with a prefix of mcd., jms., usr., or mqext.	All optional message properties (where the Support value is MQPD_SUPPORT_OPTIONAL), except those in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.

Message properties	Result
The message does not contain a property with a prefix of mcd., jms., usr., or mqext.	All message properties, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the content='properties' attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE
All properties of the message, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

RCVEXIT (string)
Receive exit name.

RCVDATA (string)
Receive exit user data.

RCVTIME(integer)
Maximum TCP/IP Wait.

SCYEXIT (string)
Security exit name.

SCYDATA (string)
Security exit user data.

SENDEXIT (string)
Send exit name.

SENDDATA (string)
Send exit user data.

SEQWRAP(integer)
Sequence number wrap.

SHORTRTY(integer)
Short connection retry count.

SHORTTMR(integer)
Short connection retry interval.

SSLCIPH(string)
SSL cipher specification.

SSLCAUTH(REQUIRED/OPTIONAL)
SSL client authentication.

SSLPEER(string)
SSL peer name.

STATCHL(QMGR/OFF/LOW/MEDIUM/HIGH)
Channel statistics setting.

TPNAME(string)
Transaction program name.

TRPTYPE(LU62/TCP)
Transport (transmission protocol) type.

XMITQ(string)
Transmission queue name.

DEFINE CHANNEL

Purpose

Use DEFINE CHANNEL to define a new channel to the queue manager.

Synonym

```
DEF CHL
```

Syntax

```
DEFINE CHANNEL(channel-name) CHLTYPE(channel-type)
TRPTYPE(trptype) optional-parameters
```

Parameters

channel-name
Channel name. The channel-name value should be unique; it should not match a channel name already defined to the queue manager.

channel-type
Channel type. The channel-type value should be the required channel type for the new channel. Valid types include:

RCVR Receiver.

RQSTR
Requester.

SDR Sender.

SVR Server.

SVRCONN
Server connection (used by clients).

trptype
Transport (transmission protocol) type.The trptype value should identify the required transport type for the new channel. Valid types include:

LU62 APPC LU 6.2 protocol.

TCP Transmission Control protocol.

optional-parameters
optional-parameters for the DEFINE CHANNEL command include:

BATCHINT(integer)
Batch interval.

BATCHSZ(integer)
Batch size.

CONNAME(string)
Connection name. The conname value should be the name of an LU 6.2 connection, or for TCP/IP sender channels, a remote hostname or IP address.

For TCP/IP sender channels, the remote port number can be appended (in parentheses) to the CONNAME. For example:

```
CONNAME('my.remote.host(1414)')
```

If the port number is not appended to the connection name, the queue manager uses the value specified by the PORTNUM parameter.

DESCR(string)
Description.

CONVERT(NO/YES)
Whether sender should convert application data.

DISCINT(integer)
Disconnection interval.

DISCRTY(integer)
Disconnection retry count.

LONGRTY(integer)
Long connection retry count.

LONGTMR(integer)
Long connection retry interval.

MAXMSGL(integer)
Maximum message length.

MAXXMIT(integer)
Maximum transmission size.

MONC(QMGR/OFF/LOW/MEDIUM/HIGH)
Channel monitoring setting.

PORTNUM(integer)
TCP/IP port number.

MSGDATA (string)
Message exit user data.

MSGEXIT (string)
Message exit name.

PROPCTL(COMPAT/ALL/NONE/)
Property control attribute.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender and Server channels and is optional. Permitted values are:

COMPAT
This is the default value.

Message properties	Result
The message contains a property with a prefix of mcd., jms., usr., or mqext.	All optional message properties (where the Support value is MQPD_SUPPORT_OPTIONAL), except those in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of mcd., jms., usr., or mqext.	All message properties, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the content='properties' attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE
All properties of the message, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

RCVEXIT (string)
Receive exit name.

RCVDATA (string)
Receive exit user data.

RCVTIME(integer)
Maximum TCP/IP Wait.

SCYEXIT (string)
Security exit name.

SCYDATA (string)
Security exit user data.

SENDEXIT (string)
Send exit name.

SENDDATA (string)
Send exit user data.

SEQWRAP(integer)
Sequence number wrap.

SHORTRTY(integer)
Short connection retry count.

SHORTTMR(integer)
Short connection retry interval.

SSLCIPH(string)
SSL cipher specification.

SSLCAUTH(REQUIRED/OPTIONAL)
SSL client authentication.

SSLPEER(string)
SSL peer name.

STATCHL(QMGR/OFF/LOW/MEDIUM/HIGH)
Channel statistics setting.

TPNAME(string)
Transaction program name.

XMITQ(string)
Transmission queue name.

DELETE CHANNEL

Purpose

Use DELETE CHANNEL to delete a channel definition.

Synonym

```
DELETE CHL
```

Syntax

```
DELETE CHANNEL(channel-name)
```

Parameters

channel-name
Channel name. The channel-name value should match an existing channel defined to the queue manager.

DISPLAY CHANNEL

Purpose

Use DISPLAY CHANNEL to display a channel definition.

Synonym

```
DIS CHL
```

Syntax

```
DISPLAY CHANNEL(channel-name) WHERE (FilterCondition) requested-attributes
```

Parameters

channel-name
Channel name. The channel-name value should match an existing channel defined to the queue manager.

The channel-name can be generic. A trailing asterisk (*) matches all channels with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channels. Characters after the first asterisk, if present, are ignored.

FilterCondition
Specify a filter condition to display only those channels that satisfy the selection criterion of the filter condition.

Almost any parameter that can be used to display attributes for this DISPLAY command. You cannot use CHLTYPE if it is also used to select channels. Channels of a type for which the filter keyword is not a valid attribute are not displayed.

Examples:

```
DIS CHL(ABC*)     WHERE(DESCR GE 'REQUESTER channel')
DIS CHL(ABC*)     WHERE(DESCR LK REQ*)
DIS CHL(ABC*)     WHERE(ALTDATE LK  2010-08*) ALTTIME
DIS CHL(FILTERS*) WHERE(ALTTIME LT  13.49.57)
DIS CHL(ABC.SDR*) WHERE(XMITQ   LK  ABC.XQ*)
DIS CHL(LU.SDR*)  WHERE(TPNAME  GT  XYZ)
DIS CHL(ABC*)     WHERE(CONNAME  LK  '1.2.3.4*')
DIS CHL(XYZ*)     WHERE(SSLPEER  NE  C=AU)
DIS CHL(XYZ*)     WHERE(SSLCIPH  LK  TRIPLE_DES*)
DIS CHL(ABC*)     WHERE(SCYEXIT  EQ  MYPROG1)
DIS CHL(ABC*)     WHERE(SCYDATA  NE  'MYPROG2 DATA')
DIS CHL(ABC.SDR*) WHERE(SENDEXIT  CT  MYPROG2)
DIS CHL(ABC.RCVR*) WHERE(RCVEXIT  CTG MYRECX2*) DESCR ALTDATE
DIS CHL(ABC*)      WHERE(MSGEXIT  EX  EXITCCC)  DESCR
DIS CHL(ABC*)      WHERE(SENDDATA  EX  'EXITABC DATA')
DIS CHL(ABC*)      WHERE(RCVDATA  EXG  REXITX*) ALTDATE ALTTIME
DIS CHL(FILTERS*)  WHERE(MSGDATA  CT  'MEXITZZZ DATA') ALTDATE
DIS CHL(*)         WHERE(CHLTYPE  EQ  RCVR)
DIS CHL(*)         WHERE(TRPTYPE NE TCP) ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(MAXMSGL  GT  3000) ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(MAXXMIT  GE  5000) DESCR
DIS CHL(*)         WHERE(RCVTIME  NE  300) ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(BATCHSZ  GT  50) DESCR ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(BATCHINT LT  20) DESCR ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(DISCRTY  LE  2)
DIS CHL(ABC*)      WHERE(DISCINT  EQ  300)
DIS CHL(*)         WHERE(SHORTRTY  GT  31) DESCR LONGRTY
DIS CHL(*)         WHERE(SHORTTMR  NE  32) DESCR LONGTMR
DIS CHL(*)         WHERE(LONGRTY  EQ  32) ALTDATE ALTTIME
DIS CHL(*)         WHERE(LONGTMR  EQ  320)
DIS CHL(ABC*)      WHERE(SEQWRAP  NE  99999999)DESCR
DIS CHL(ABC*)      WHERE(CONVERT  NE  YES) ALTDATE ALTTIME
DIS CHL(ABC*)      WHERE(SSLCAUTH  EQ REQUIRED)
DIS CHL(ABC*)      WHERE(STATCHL  EQ  MEDIUM)
DIS CHL(ABC*)      WHERE(MONCHL  EQ  HIGH)  DESCR
```

requested-attributes

Attributes of the channel that are to be displayed. This can be:

ALL Displays all channel attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

BATCHINT(integer)
Batch interval.

BATCHSZ
Batch size.

DESCR
Channel description.

CHLTYPE
Channel type.

CONNAME
Connection name.

CONVERT
Whether sender should convert application data.

DISCINT
Disconnection interval.

DISCRTY
Disconnection retry count.

LONGRTY(integer)
Long connection retry count.

LONGTMR(integer)
Long connection retry interval.

MAXMSGL
Maximum message length.

MAXXMIT(integer)
Maximum transmission size.

MONC
Channel monitoring setting.

MSGEXIT
Message exit name.

MSGDATA
Message exit user data.

PORTNUM
TCP/IP port number.

PROPCTL
Message property control.

RCVEXIT
Receive exit name.

RCVDATA
Receive exit user data.

RCVTIME(integer)
Maximum TCP/IP Wait.

SCYEXIT
Security exit name.

SCYDATA
Security exit user data.

SENDEXIT
Send exit name.

SENDDATA
Send exit user data.

SEQWRAP
Sequence number wrap.

SHORTRTY(integer)
Short connection retry count.

SHORTTMR(integer)
Short connection retry interval.

SSLCIPH
SSL cipher specification.

SSLCAUTH
SSL client authentication.

SSLPEER
SSL peer name.

STATC
Channel statistics setting.

TPNAME
Transaction program name.

TRPTYPE
Transport (transmission protocol) type.

XMITQ
Transmission queue name.

DISPLAY CHSTATUS

Purpose

Use the MQSC command DISPLAY CHSTATUS to display the status of one or more channels.

Synonym

```
DIS CHS
```

Syntax

```
DISPLAY   CHSTATUS (generic-channel-name)
WHERE (FilterCondition) requested-attributes
```

Usage

You must specify the name of the channel for which you want to display status information. This can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

You can also specify whether you want the current status data (of current channels only), or the saved status data of all channels.

Status for all channels that meet the selection criteria is given, whether the channels were defined manually or automatically.

There are two classes of data available for channel status. These are saved and current.

The status fields available for saved data are a subset of the fields available for current data and are called common status fields. Note that although the common data fields are the same, the data values might be different for saved and current status. The rest of the fields available for current data are called current-only status fields.

- Saved data consists of the common status fields. This data is reset for all channels when the channel enters or leaves STOPPED or RETRY state. For a sending channel data is reset before requesting confirmation that a batch of messages has been received and when confirmation has been received. For a receiving channel data is reset just before confirming that a batch of messages

has been received. For a server connection channel no data is saved. Therefore, a channel that has never been current cannot have any saved status. Note: Because status is saved at the end of each batch, a channel does not have any saved status until at least one batch has been transmitted.

- Current data consists of the common status fields and current-only status fields as noted in the syntax diagram. The data fields are continually updated as messages are sent/received.

This method of operation has the following consequences:

- An inactive channel might not have any saved status if it has never been current or has not yet reached a point where saved status is reset.
- The "common" data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can either be current or inactive:

Current channels
These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They might not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have current status and might also have saved status.

The term Active is used to describe the set of current channels that are not stopped.

Inactive channels
These are channels that either:

- Have not been started.
- On which a client has not connected.
- Have finished.
- Have disconnected normally.

Inactive channels have either saved status or no status at all.

There can be more than one instance of the same server-connection channel current at the same time. For channels of other types, there can only be one instance current at any time.

Parameters

(generic-channel-name)
The name of the channel definition for which status information is to be displayed. A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions.

ALL Specify this to display all the status information for each relevant instance.

If SAVED is specified, this causes only common status information to be displayed, not current-only status information.

If this parameter is specified, any parameters requesting specific status information that are also specified have no effect; all the information is displayed.

CONNAME(connection-name)
The connection name for which status information is to be displayed, for the specified channel or channels.

This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The value returned for CONNAME might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using CONNAME for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if CONNAME in the channel definition:
- Is blank or is in "host name" format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number (except on z/OS), but the saved channel status value does not.

For SAVED status, this value could also be the queue manager name of the remote system.

CURRENT
This is the default, and indicates that current status information as held by the channel initiator for current channels only is to be displayed.

Both common and current-only status information can be requested for current channels.

SAVED
Specify this to display saved status information for both current and inactive channels.

Only common status information can be displayed. Current-only status information is not displayed for current channels if this parameter is specified.

WHERE
Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The parameter to be used to display attributes for this DISPLAY command. However, you cannot use the following parameters as filter keywords: COMPRATE, COMPTIME,CURRENT, EXITTIME, MONITOR, NETTIME, SAVED, SHORT, XBATCHSZ, or XQTIME.

You cannot use CONNAME or XMITQ as filter keywords if you also use them to select channel status.

Status information for channels of a type for which the filter keyword is not valid is not displayed.

Examples:

```
DISPLAY CHSTATUS(ABC*)  SAVED    WHERE(CHLTYPE EQ RCVR)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(CONNAME LT '4.2.3.4(1417)')
DISPLAY CHSTATUS(*)     ALL      WHERE(CURLUWID LT 0000000000000000)
DISPLAY CHSTATUS(ABC*)  SAVED    WHERE(LSTLUWID LK '00*')
DISPLAY CHSTATUS(ABC*)  SAVED    WHERE(XMITQ NL 'ABC.XQ*')
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(CURMSGS EQ 10)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(CURSEQNO GT 100000)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(INDOUBT EQ NO)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(LSTSEQNO GE 999999)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(STATUS EQ RUNNING)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(SUBSTATE EQ MQGET)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(RQMNAME LK TEST*)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(BATCHES GT 10)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(BATCHSZ GE 10)
```

```
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(BUFSRCVD LE 1000)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(BUFSSEND GT 1000)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(BYTSRCVD LT 10000)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(BYTSSEND LT 10240)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(CHSTADA EQ 2010-09-14)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(CHSTATI NL 05.*)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(LOCLADDR EQ '1.2.3.4(4212)')
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(SHORTRTS LE 12)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(LONGRTS  GT 21)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(LSTMSGDA GT 2010-08-18)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(LSTMSGTI NL 05.*)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(MCAUSER LK XX*)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(MONCHL NE HIGH)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(MSGS GE 15)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(SSLCERTI LK 'CN=IBM*')
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(SSLKEYDA GT 2010-08-18)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(SSLKEYTI GT 09.47.05)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(SSLPEER LK C=AU*)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(SSLRKEYS LE 10)
DISPLAY CHSTATUS(*)     CURRENT  WHERE(STOPREQ  NE NO)
DISPLAY CHSTATUS(ABC*)  CURRENT  WHERE(XMSGSA GT 5)
```

MONITOR

Specify this to return the set of online monitoring parameters. These are COMPRATE, COMPTIME, EXITTIME, MONCHL, NETTIME, XBATCHSZ, XQMSGSA, and XQTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

XMITQ(q-name)

The name of the transmission queue for which status information is to be displayed, for the specified channel or channels. This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The following information is always returned, for each set of status information:

- The channel name.
- The transmission queue name (for sender and server channels).
- The connection name.
- The remote queue-manager name (only for current status, and for all channel types except server-connection channels).
- The type of status information returned (CURRENT or SAVED).
- STATUS.
- STOPREQ (only for current status).
- SUBSTATE.

If no parameters requesting specific status information are specified (and the ALL parameter is not specified), no further information is returned.

If status information is requested that is not relevant for the particular channel type, this is not an error.

Common status

The following information applies to all sets of channel status, whether or not the set is current. The information applies to all channel types except server-connection.

CHLTYPE

The channel type. This is one of the following:

SDR A sender channel

SVR A server channel
RCVR A receiver channel
RQSTR
A requester channel
SVRCONN
A server-connection channel

CURLUWID
The logical unit of work identifier associated with the current batch, for a sending or a receiving channel. For a sending channel, when the channel is in doubt it is the LUWID of the in-doubt batch. For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt. It is updated with the LUWID of the next batch when this is known.

CURMSGS
For a sending channel, this is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in doubt it is the number of messages that are in doubt. For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt. For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received. The value is reset to zero, for both sending and receiving channels, when the batch is committed.

CURSEQNO
For a sending channel, this is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in doubt it is the message sequence number of the last message in the in-doubt batch. For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt. For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

INDOUBT
Whether the channel is currently in doubt. This is only YES while the sending Message Channel Agent is waiting for an acknowledgement that a batch of messages that it has sent has been successfully received. It is NO at all other times, including the period during which messages are being sent, but before an acknowledgement has been requested. For a receiving channel, the value is always NO.

LSTLUWID
The logical unit of work identifier associated with the last committed batch of messages transferred.

LSTSEQNO
Message sequence number of the last message in the last committed batch. This number is not incremented by nonpersistent messages using channels with a NPMSPEED of FAST.

STATUS
Current status of the channel. This is one of the following:

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

INITIALIZING
The channel initiator is attempting to start a channel.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
Channel is stopping or a close request has been received.

RETRYING
A previous attempt to establish a connection has failed. The MCA will reattempt connection after the specified time interval.

PAUSED
The channel is waiting for the message-retry interval to complete before retrying an MQPUT operation.

STOPPED
This state can be caused by one of the following:

- Channel manually stopped A user has entered a stop channel command against this channel.
- Retry limit reached The MCA has reached the limit of retry attempts at establishing a connection. No further attempt will be made to establish a connection automatically.

REQUESTING
A local requester channel is requesting services from a remote MCA.

Note: For an inactive channel, CURMSGS, CURSEQNO, and CURLUWID have meaningful information only if the channel is INDOUBT. However they are still displayed and returned if requested.

Current-only status

Parameter descriptions for the current channel instances of the DISPLAY CHSTATUS command.

The following information applies only to current channel instances. The information applies to all channel types, except where stated.

BATCHES
Number of completed batches during this session (since the channel was started).

BATCHSZ
The batch size being used for this session. The batch size being used for this session. This parameter does not apply to server-connection channels, and no values are returned; if specified on the command, this is ignored.

BUFSRCVD
Number of transmission buffers received. This includes transmissions to receive control information only.

BUFSSENT
Number of transmission buffers sent. This includes transmissions to send control information only.

BYTSRCVD
Number of bytes received during this session (since the channel was started). This includes control information received by the message channel agent.

BYTSSENT
Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

CHSTADA
Date when this channel was started (in the form yyyy-mm-dd).

CHSTATI
Time when this channel was started (in the form hh.mm.ss).

CURSHCNV
The CURSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the CURSHCNV output gives a count of the number of conversations currently running over that channel instance.

EXITTIME
Amount of time, displayed in microseconds, spent processing user exits per message. Two values are displayed: A value based on recent activity over a short period of time. A value based on activity over a longer period of time. These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONCHL is set for this channel.

LOCLADDR
Local communications address for the channel. The value returned depends on the TRPRYPE of the channel (currently only TCP/IP is supported).

LONGRTS
Number of long retry wait start attempts left. This applies only to sender or server channels.

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. In the case of a

server-connection channel instance on which conversations are being shared, this is the time when the last MQI call completed on any of the conversations running on the channel instance.

MAXSHCNV
The MAXSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the MAXSHCNV output gives the negotiated maximum of the number of conversations that can run over that channel instance.

MCASTAT
Whether the Message Channel Agent is currently running. This is either "running" or "not running". Note that it is possible for a channel to be in stopped state, but for the program still to be running.

MCAUSER
The user ID used by the MCA. This can be the user ID set in the channel definition, the default user ID for MCA channels, a user ID transferred from a client if this is a server-connection channel, or a user ID specified by a security exit. This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels. On server connection channels that share conversations, the MCAUSER field contains a user ID if all the conversations have the same MCA user ID value. If the MCA user ID in use varies across these conversations, the MCAUSER field contains a value of *. The maximum length is 12 characters on z/OS; on other platforms, it is 64 characters.

MONCHL
Current level of monitoring data collection for the channel. This parameter is also displayed when you specify the MONITOR parameter.

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). In the case of a server-connection channel instance on which conversations are being shared, this is the total number of MQI calls handled on all of the conversations running on the channel instance.

NETTIME
Amount of time, displayed in microseconds, to send a request to the remote end of the channel and receive a response. Two values are displayed:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. This parameter applies only to sender, server, and cluster-sender channels. This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONCHL is set for this channel.

RQMNAME
The queue manager name, or queue-sharing group name, of the remote system. This parameter does not apply to server-connection channels.

SHORTRTS
Number of short retry wait start attempts left. This applies only to sender or server channels.

SSLCERTI
The full Distinguished Name of the issuer of the remote certificate. The issuer is the Certificate Authority that issued the certificate. The maximum length is 256 characters. This limit might mean that exceptionally long Distinguished Names are truncated.

SSLKEYDA
Date on which the previous successful SSL secret key reset was issued.

SSLKEYTI
Time at which the previous successful SSL secret key reset was issued.

SSLPEER
Distinguished Name of the peer queue manager or client at the other end of the channel. The maximum length is 256 characters. This limit might mean that exceptionally long Distinguished Names are truncated.

SSLRKEYS
Number of successful SSL key resets. The count of SSL secret key resets is reset when the channel instance ends.

STOPREQ
Whether a user stop request is outstanding. This is either YES or NO.

SUBSTATE
Action being performed by the channel when this command is issued. The following substates are listed in precedence order, starting with the substate of the highest precedence:

ENDBATCH
Channel is performing end-of-batch processing.

SEND A request has been made to the underlying communication subsystem to send some data.

RECEIVE
A request has been made to the underlying communication subsystem to receive some data.

RESYNCH
Channel is resynchronizing with the partner.

HEARTBEAT
Channel is heartbeating with the partner.

SCYEXIT
Channel is running the security exit.

RCVEXIT
Channel is running one of the receive exits.

SENDEXIT
Channel is running one of the send exits.

MSGEXIT
Channel is running one of the message exits.

MREXIT
Channel is running the message retry exit.

CHADEXIT
Channel is running through the channel auto-definition exit.

NETCONNECT
A request has been made to the underlying communication subsystem to connect a partner machine.

SSLHANDSHK
Channel is processing an SSL handshake.
NAMESERVER
A request has been made to the name server.
MQPUT
A request has been made to the queue manager to put a message on the destination queue.
MQGET
A request has been made to the queue manager to get a message from the transmission queue (if this is an MCA channel) or from an application queue (if this is an MQI channel).
MQICALL
A MQ API call, other than MQPUT and MQGET, is being executed.

Not all substates are valid for all channel types or channel states. There are occasions when no substate is valid, at which times a blank value is returned. For channels running on multiple threads, this parameter displays the substate of the highest precedence.

XBATCHSZ
Size of the batches transmitted over the channel. Two values are displayed:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. This parameter does not apply to server-connection channels. This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONCHL is set for this channel.

XQMSGSA
Number of messages queued on the transmission queue available to the channel for MQGETs. This parameter has a maximum displayable value of 999. If the number of messages available exceeds 999, a value of 999 is displayed. On z/OS, if the transmission queue is not indexed by CorrelId, this value is shown as blank. This parameter applies to cluster-sender channels only. This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONCHL is set for this channel.

XQTIME
The time, in microseconds, that messages remained on the transmission queue before being retrieved. The time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application. Two values are displayed: A value based on recent activity over a short period of time. A value based on activity over a longer period of time. These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. This parameter applies only to sender, server, and cluster-sender channels. This parameter is also displayed when you

specify the MONITOR parameter. A value is only displayed for this parameter if MONCHL is set for this channel.

RESET CHANNEL
Purpose

Use RESET CHANNEL to reset the message sequence number for an WebSphere MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

Synonym

```
RESET CHL
```

Syntax

```
RESET CHANNEL(channel-name) optional-parameter
```

Parameters

channel-name
Channel name. The channel-name value should match an existing channel defined to the queue manager.

optional-parameter
The option parameter for RESET CHANNEL command is as follows:

SEQNUM(integer)
The new message sequence number, which must be greater than or equal to 1, and less than or equal to 999 999 999. If this parameter is not specified, the sequence number is reset to 1.

START CHANNEL
Purpose

Use START CHANNEL to start a channel.

Synonym

```
STA CHL
```

Syntax

```
START CHANNEL(channel-name)
```

Parameters

channel-name
Channel name. The channel-name value should match an existing channel defined to the queue manager.

STOP CHANNEL
Purpose

Use STOP CHANNEL to stop a channel.

Synonym

```
STOP CHL
```

Syntax

```
STOP CHANNEL(channel-name) optional-parameter
```

Parameters

channel-name
Channel name. The channel-name value should match an existing channel defined to the queue manager.

optional-parameter
The option parameter for STOP CHANNEL command is:

STATUS(INACTIVE/STOPPED)
Indicates whether the channel should be stopped and placed in INACTIVE or STOPPED state.

WebSphere MQ channel authentication

The WebSphere MQ channel authentication commands are:
DISPLAY CHLAUTH
SET CHLAUTH

DISPLAY CHLAUTH

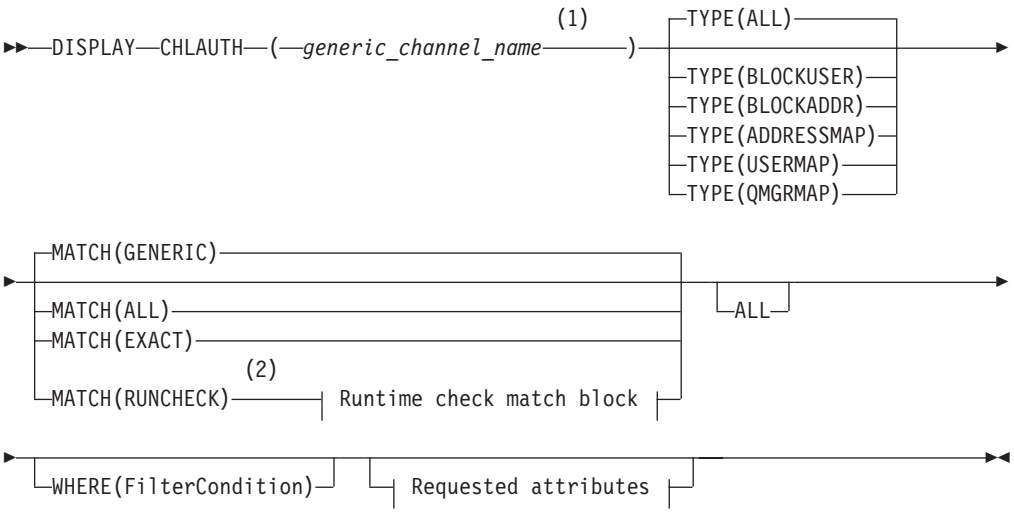
Purpose

Use the MQSC command DISPLAY CHLAUTH to display the attributes of a channel authentication record.

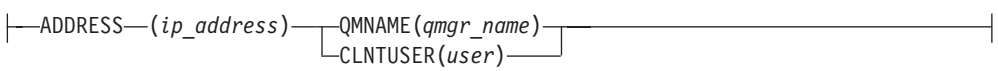
Synonym

DIS CHLAUTH

Syntax



Runtime check match block:



Requested attributes:

Notes:

1 Must be * with TYPE(BLOCKADDR) and cannot be generic with MATCH(RUNCHECK).

2 Must be combined with TYPE(ALL).

Parameters

generic-channel-name
The name of the channel or set of channels to display. You can use the asterisk (*) as a wildcard to specify a set of channels. When MATCH is RUNCHECK this parameter must not be generic.

ADDRESS
The IP address to be matched.

This parameter is valid only when MATCH is RUNCHECK and must not be generic.

ALL Specify this parameter to display all attributes. If this keyword is specified, any attributes that are requested specifically have no effect; all attributes are still displayed. This is the default behavior if you do not specify a generic name and do not request any specific attributes.

CLNTUSER
The client user ID to be matched.

This parameter is valid only when MATCH is RUNCHECK and must not be generic.

MATCH
Indicates the type of matching to be applied.

RUNCHECK
Returns the record that will be matched by a specific inbound channel at run time if it connects into this queue manager. The specific inbound channel is described by providing values that are not generic for:
- The channel name
- ADDRESS attribute
- QMNAME or CLNTUSER attribute, depending on whether the inbound channel will be a client or queue manager channel

If the record discovered has WARN set to YES, a second record might also be displayed to show the actual record the channel will use at runtime. This parameter must be combined with TYPE(ALL).

EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MATCH(GENERIC).

GENERIC
Any asterisks in the channel profile name are treated as wild cards. If there are no asterisks in the channel profile name, this returns the same output as MATCH(EXACT). For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

ALL Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more specific matches exist.

QMNAME
The name of the remote partner queue manager to be matched This parameter is valid only when MATCH is RUNCHECK and must not be generic.

TYPE The type of Channel Authentication Record for which to display details. Possible values are:
- ALL
- BLOCKUSER
- BLOCKADDR
- ADDRESSMAP
- USERMAP
- QMGRMAP

WHERE
See "WHERE" on page 511 for information about this parameter.

Requested attributes

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

TYPE The type of channel authentication record

ADDRESS
The IP address

CLNTUSER
The client asserted user ID

QMNAME
The name of the remote partner queue manager

MCAUSER
The user identifier to be used when the inbound connection matches the IP address, client asserted user ID or remote queue manager name supplied.

ADDRLIST
A list of IP address patterns which are banned from connecting into this queue manager on any channel.

USERLIST
A list of user IDs which are banned from use of this channel or set of channels.

ALTDATE
The date on which the channel authentication record was last altered, in the format yyyy-mm-dd.

ALTTIME
The time on which the channel authentication record was last altered, in the form hh.mm.ss.

DESCR
Descriptive information about the channel authentication record.

SET CHLAUTH

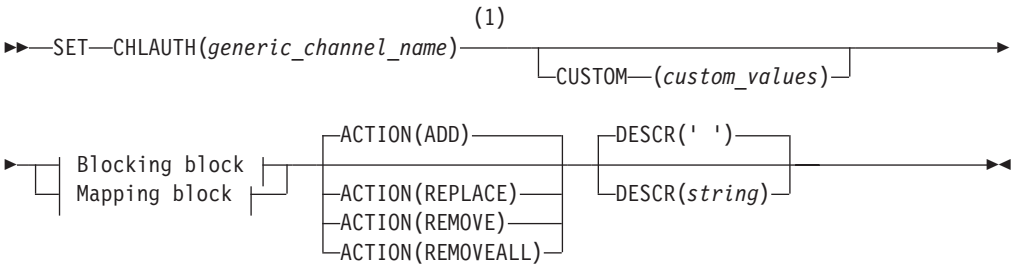
Purpose

Use the MQSC command SET CHLAUTH to create or modify a channel authentication record.

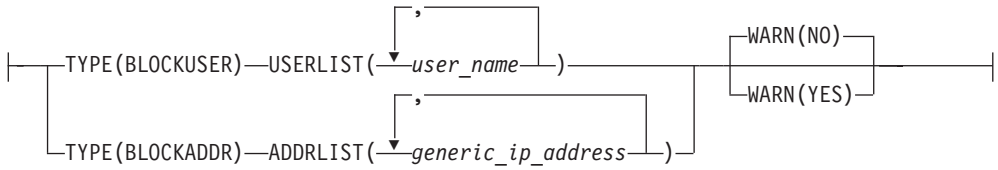
Synonym

DIS CHLAUTH

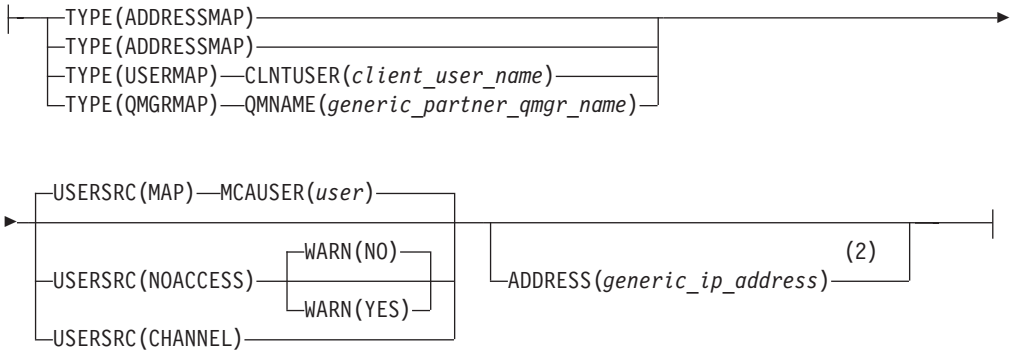
Syntax



Blocking block:



Mapping block:



Notes:

1 The generic channel name must be '*' when TYPE is BLOCKADDR.

2 Mandatory when TYPE is ADDRESSMAP.

Usage notes

The following table shows which parameters are valid for each value of ACTION:

Table 12. Valid actions

Parameter	ADD or REPLACE	REMOVE	REMOVEALL
CHLAUTH	Y	Y	Y
TYPE	Y	Y	Y
ACTION	Y	Y	Y
ADDRESS	Y	Y	
ADDRLIST	Y	Y	
CLNTUSER	Y	Y	
MCAUSER	Y		
QMNAME	Y	Y	
USERLIST	Y	Y	
USERSRC	Y		
WARN	Y		
DESCR	Y		

Parameters

generic-channel-name
The name of the channel or set of channels for which you are setting channel authentication configuration. You can use a trailing asterisk (*) as a wildcard to specify a set of channels. If you set TYPE to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

ACTION
The action to perform on the channel authentication record. The following values are valid:

ADD Add the specified configuration to a channel authentication record. This is the default value. For types ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, the command fails. For types BLOCKUSER and BLOCKADDR, the configuration is added to the list.

REPLACE
Replace the current configuration of a channel authentication record. For types ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added. For types BLOCKUSER and BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like REMOVEALL.

REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like REMOVEALL.

REMOVEALL
Remove all members of the list and thus the whole record (for BLOCKADDR and BLOCKUSER) or all previously defined

mappings (for ADDRESSMAP, QMGRMAP and USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in ADDRLIST, USERLIST, ADDRESS, QMNAME or CLNTUSER. If the specified type has no current configuration the command still succeeds.

ADDRESS
The filter to be used to compare with the IP address of the partner queue manager or client at the other end of the channel. This parameter is mandatory with TYPE(ADDRESSMAP) This parameter is also valid when TYPE is USERMAP or QMGRMAP and ACTION is ADD, REPLACE, or REMOVE. You can define more than one channel authentication object with the same main identity, for example a queue manager map with the same remote queue manager name, with different addresses. However, you cannot define channel authentication records with overlapping address ranges for the same main identity.

ADDRLIST
A list of up to 56 generic IP addresses which are banned from accessing this queue manager on any channel. This parameter is only valid with TYPE(BLOCKADDR).

CLNTUSER
The client asserted user ID to be mapped to a new user ID or blocked. This parameter is valid only with TYPE(USERMAP).

DESCR
Provides descriptive information about the channel authentication record, which is displayed when you issue the DISPLAY CHLAUTH command. It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes). Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUSER
The user identifier to be used when the inbound connection matches the IP address, client asserted user ID or remote queue manager name supplied. This parameter is mandatory with USERSRC(MAP) and is valid when TYPE is ADDRESSMAP, USERMAP, or QMGRMAP. This parameter can only be used when ACTION is ADD or REPLACE.

QMNAME
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked. This parameter is valid only with TYPE(QMGRMAP).

TYPE The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER. This parameter is required. The following values can be used:

BLOCKUSER
This channel authentication record prevents a specified user or users from connecting. The BLOCKUSER parameter must be accompanied by a USERLIST.

BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses. The BLOCKADDR parameter must be accompanied by an ADDRLIST. BLOCKADDR operates at the listener before the channel name is known.

ADDRESSMAP
This channel authentication record maps IP addresses to

MCAUSER values. The ADDRESSMAP parameter must be accompanied by an ADDRESS. ADDRESSMAP operates at the channel.

USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The USERMAP parameter must be accompanied by a CLNTUSER. QMGRMAP This channel authentication record maps remote queue manager names to MCAUSER values. The QMGRMAP parameter must be accompanied by a QMNAME.

USERLIST
A list of up to 100 user IDs which are banned from use of this channel or set of channels. This parameter is only valid with TYPE(BLOCKUSER).

USERSRC
The source of the user ID to be used for MCAUSER at run time. The following values are valid:

MAP Inbound connections that match this mapping use the user ID specified in the MCAUSER attribute.

NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Note that WARN and USERSRC(CHANNEL), or USERSRC(MAP) are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

WARN
Indicates whether this record operates in warning mode.

NO This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

YES This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if channel events are configured, a channel event message is created showing the details of what would have been blocked, The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Error codes

This command might return the following error codes in the response format header.

Reason (MQLONG)
The value can be:

MQRCCF_CHLAUTH_TYPE_ERROR
Channel authentication record type not valid.

MQRCCF_CHLAUTH_ACTION_ERROR
Channel authentication record action not valid.

MQRCCF_CHLAUTH_USERSRC_ERROR
Channel authentication record user source not valid.

MQRCCF_WRONG_CHLAUTH_TYPE
Parameter not allowed for this channel authentication record type.

MQRCCF_CHLAUTH_ALREADY_EXISTS
Channel authentication record already exists

WebSphere MQ channel listener

Listener objects can be created, modified, deleted and displayed using MQSC commands. The following MQSC commands are supported:

ALTER LISTENER
DEFINE LISTENER
DELETE LISTENER
DISPLAY LISTENER
DISPLAY LSSTATUS
START LISTENER
STOP LISTENER

ALTER LISTENER

Purpose

Alter the parameters of an existing WebSphere MQ channel listener definition. Listener must be in STOPPED status in order to modify any attributes.

Synonym

```
ALT LSTR
```

Syntax

Parameters

listener-name
Name of the WebSphere MQ channel listener definition to alter.

Optional-parameters
For details of optional parameters for the ALTER LISTENER command, see "DEFINE LISTENER."

DEFINE LISTENER

Purpose

Define a new WebSphere MQ channel listener definition, and set its parameters.

Synonym

```
DEF LSTR
```

Syntax

```
DEFINE LISTENER(listener-name) optional-parameters
```

Parameters

listener-name
Name of the WebSphere MQ channel listener definition to alter.

Optional-parameters
Optional parameters for the DEFINE LISTENER command include:

BACKLOG(integer)
The number of concurrent connection requests that the listener supports. If you do not specify a value, the default value defined by the protocol is used.

CONTROL(string)
Specifies how the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands. This is the default value.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

DESCR(string)
Plain-text comment. It provides descriptive information about the listener when an operator issues the DISPLAY LISTENER command (see DISPLAY LISTENER). It should contain only displayable characters.

The maximum length is 64 characters.

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

IPADDR(string)
IP address for the listener specified in IPv4 dotted decimal or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 stacks.

PORT(integer)
The port number for TCP/IP.

TRPTYPE(string)
The transmission protocol to be used:

TCP for TCP/IP

DELETE LISTENER
Purpose

Delete a channel listener definition. Listener must be in STOPPED status in order to delete the definition.

Synonym

```
DEL LSTR
```

Syntax

```
DELETE LISTENER(listener-name)
```

Parameters

listener-name
Name of the WebSphere MQ channel listener definition to delete.

DISPLAY LISTENER
Purpose

Display information about a channel listener. The values displayed describe the current definition of the listener. If the listener has been altered since it was started,

the currently running instance of the listener object may not have the same values as the current definition.

Synonym

DIS LSTR

Syntax

```
DISPLAY   LISTENER(generic-listener-name)
WHERE (FilterCondition) requested-attributes
```

Parameters

generic-listener-name
The name of the listener definition for which information is to be displayed. A single asterisk (*) specifies that information for all listener identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

Requested-attributes

TRPTYPE
Transmission protocol. If you do not specify this parameter, a default of ALL is assumed. Values are:

ALL This is the default value and displays information for all listeners.

TCP Displays information for all listeners defined with a value of TCP in their TRPTYPE parameter.

ALL Specify this to display all the listener information for each specified listener. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

ALTDATE
The date on which the definition was last altered, in the form *yyyy-mm-dd*.

ALTTIME
The time at which the definition was last altered, in the form *hh.mm.ss*.

BACKLOG
The number of concurrent connection requests that the listener supports.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

PORT The port number for TCP/IP.

Examples

```
DISPLAY LISTENER(ABC*)   WHERE(ALTDATE GT 2010-08-19)
DISPLAY LISTENER(*)      WHERE(ALTTIME LK 08.1*)
DISPLAY LISTENER(*)      WHERE(IPADDR LK 9.*)
DIS LSTR(LISTENER(*)      WHERE(DESCR LK 'TEST*') ALTDATE ALTTIME
DISPLAY LISTENER(*)      WHERE(BACKLOG GE 5)
DISPLAY LISTENER(ABC*)   WHERE(CONTROL EQ QMGR)
DISPLAY LISTENER(*) TRPTYPE(TCP) WHERE(PORT EQ 1414)
DISPLAY LISTENER(*)      WHERE(TRPTYPE EQ TCP)
```

DISPLAY LSSTATUS

Purpose

Display status information for one or more channel listeners. Only listeners with status of RUNNING are returned.

Synonym

```
DIS LSSTATUS
```

Syntax

```
DISPLAY   LSSTATUS ( generic-listener-name )
WHERE (FilterCondition) requested attrs
```

Parameters

generic-listener-name
The name of the listener definition for which information is to be displayed. A single asterisk (*) specifies that information for all listener identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

Requested-attributes

ALL Specify this to display all the listener information for each specified listener. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

BACKLOG
The number of concurrent connection requests that the listener supports.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

PORT The port number for TCP/IP.

STARTDA
The date on which the listener was started.

STARTTI
The time at which the listener was started.

STATUS
The current status of the listener. It can be one of:
RUNNING
The listener is running.
STOPPED
The listener is stopped.

TRPTYPE
Transport type.

PID CICS task number

Examples

```
DISPLAY LSSTATUS(ABC*)   WHERE(DESCR NL 'TEST*')
DISPLAY LSSTATUS(ABC*)   WHERE(IPADDR LK 9.*)
DISPLAY LSSTATUS(ABC*)   WHERE(STARTDA NE 2010-09-14)
DISPLAY LSSTATUS(*)      WHERE(STARTTI GE 17.09.30)
DISPLAY LSSTATUS(ABC*)   WHERE(BACKLOG EQ 5)
DISPLAY LSSTATUS(ABC*)   WHERE(CONTROL EQ MANUAL)
DISPLAY LSSTATUS(*)      WHERE(PORT NE 1421)
DISPLAY LSSTATUS(ABC*)   WHERE(TRPTYPE EQ TCP)
DISPLAY LSSTATUS(*)      WHERE(PID EQ 294)
DISPLAY LSSTATUS(ABC*)   WHERE(STATUS NE RUNNING)
```

START LISTENER

Purpose

Start a channel listener.

Synonym

```
STA LSTR
```

Syntax

```
START LISTENER(listener-name)
```

Parameters

listener-name
Name of the channel listener to be started.

STOP LISTENER
Purpose

Stop a channel listener.

Synonym

```
STOP LSTR
```

Syntax

```
STOP LISTENER(listener-name)
```

Parameters

listener-name
Name of the channel listener to be stopped.

WebSphere MQ connection commands

The WebSphere MQ connection commands are:
DISPLAY CONN
STOP CONN

DISPLAY CONN
Purpose

Use the MQSC command DISPLAY CONN to display connection information about the applications connected to the queue manager. This is a useful command because it enables you to identify applications with long-running units of work.

Synonym

```
DIS CONN
```

Syntax

```
DISPLAY   CONN(generic-connid)
WHERE(FilterCondition) EXTCONN(extconn) requested-attributes
```

Parameters

You must specify a connection for which you want to display information. This can be a specific connection identifier or a generic connection identifier. A single asterisk (*) can be used as a generic connection identifier to display information for all connections.

(generic-connid)
The identifier of the connection definition for which information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed.

When an application connects to WebSphere MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value for CONN is formed by converting the last eight bytes of the ConnectionId to its 16-character hexadecimal equivalent.

ALL Specify this to display all the connection information of the requested type for each specified connection. This is the default if you do not specify a generic identifier, and do not request any specific parameters.

EXTCONN
The value for EXTCONN is based on the first 16 bytes of the ConnectionId converted to its 32-character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the EXTCONN parameter. Use the CONN parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both EXTCONN and CONN) to avoid possible problems related to non-unique CONN values.

Do not specify both a generic value for CONN and a non-generic value for EXTCONN.

TYPE Specifies the type of information to be displayed. Values are:

CONN
Connection information for the specified connection.

HANDLE
Information relating to any objects opened by the specified connection.

ALL Display all available information relating to the connection.

Connection attributes

If TYPE is set to CONN, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:

- Connection identifier (CONN parameter)
- Type of information returned (TYPE parameter)

The following parameters can be specified for TYPE(CONN) to request additional information for each connection. If a parameter is specified that is not relevant for the connection, operating environment, or type of information requested, that parameter is ignored.

APPLTAG(string)
A string containing the tag of the application connected to the queue manager. It is one of the following:

- CICS APPLID

APPLTYPE(integer)
A string indicating the type of the application that is connected to the queue manager. It is one of the following:

BATCH
Application using a batch connection.

CICS CICS transaction.

CHANNEL(string)
The name of the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CONNAME(string)
The connection name associated with the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CONNOPTS(integer-list)
The connect options currently in force for this application connection. Possible values are:

- MQCNO_NONE
- MQCNO_ACCOUNTING_Q_ENABLED
- MQCNO_ACCOUNTING_Q_DISABLED
- MQCNO_ACCOUNTING_MQI_ENABLED
- MQCNO_ACCOUNTING_MQI_DISABLED

TASKNO(string)
A 7-digit CICS task number is returned as a 7-byte string. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task.

TRANSID(string)
A 4-character CICS transaction identifier.

UOWSTATE(integer)
The state of the unit of work. It is one of the following:
NONE
There is no unit of work.
ACTIVE
The unit of work is active.

UOWSTDA(string)
The date that the transaction associated with the current connection was started.

UOWSTTI(string)
The time that the transaction associated with the current connection was started.

URTYPE(integer)
The type of unit of recovery as seen by the queue manager. It is one of the following:

- CICS

USERID(string)
The user identifier associated with the connection.

Handle attributes

If TYPE is set to HANDLE, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:
- Connection identifier (CONN parameter).
- Type of information returned (TYPE parameter).
- Handle status (HSTATE).
- Object name (OBJNAME parameter).
- Object type (OBJTYPE parameter).

The following parameters can be specified for TYPE(HANDLE) to request additional information for each queue. If a parameter is specified that is not relevant for the connection, operating environment, or type of status information requested, that parameter is ignored.

HSTATE(integer)
The state of the handle. Possible values are:
ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.
INACTIVE
No API call from this connection is currently in progress

for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

OBJNAME(string)
The name of an object that the connection has open.

OBJTYPE(integer)
The type of the object that the connection has open. It is one of the following:
QUEUE
Queue
QMGR
Queue manager
NAMELIST
Namelist

OPENOPTS(integer)
The open options currently in force for the connection for the object. Possible values are:
MQOO_INPUT_SHARED
Open queue to get messages with shared access.
MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.
MQOO_BROWSE
Open queue to browse messages.
MQOO_OUTPUT
Open queue to put messages.
MQOO_INQUIRE
Open queue to inquire attributes.
MQOO_SET
Open queue to set attributes.
MQOO_FAIL_IF_QUIESCING
Fail if queue manager is quiescing.

Full attributes

If TYPE is set to *, or ALL, both Connection attributes and Handle attributes are returned for each connection that satisfies the selection criteria.

Examples

```
DISPLAY CONN(*) WHERE(OPENOPTS CT MQOO_INPUT_SHARED)
DISPLAY CONN(01AC83C6006EE358) TYPE(CONN) +
               WHERE(HSTATE EQ INACTIVE)
DISPLAY CONN(*) WHERE(OBJNAME LE FILTERS.QSTATUS.XQ)
DISPLAY CONN(*) TYPE(CONN) WHERE(USERID LK SI*)
DISPLAY CONN(*) WHERE(UOWSTDA GE 2010-09-28)
DISPLAY CONN(*) WHERE(TASKNO GT 0001489)
```

STOP CONN
Purpose

Use the MQSC command STOP CONN to break a connection between an application and the queue manager.

There may be circumstances in which the queue manager cannot implement this command when the success of this command cannot be guaranteed.

Synonym

```
STOP CONN
```

Syntax

```
STOP CONN(connection-identifier) EXTCONN(extconn)
```

Parameters

connection-identifier, STOP CONN parameter
The identifier of the connection definition for the connection to be broken.

When an application connects to WebSphere MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value of CONN is formed by converting the last eight bytes of the ConnectionId to its 16-character hexadecimal equivalent.

EXTCONN
The value of EXTCONN is based on the first 16 bytes of the ConnectionId converted to its 32-character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the EXTCONN parameter. Use the CONN parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both EXTCONN and CONN) to avoid possible problems related to non-unique CONN values.

WebSphere MQ namelist commands

The WebSphere MQ namelist commands are:

- ALTER NAMELIST
- DEFINE NAMELIST
- DELETE NAMELIST
- DISPLAY NAMELIST

ALTER NAMELIST
Purpose

Use ALTER NAMELIST to alter the parameters of a namelist object.

Synonym

```
ALT NL
```

Syntax

```
ALTER NAMELIST (namelist-name) optional parameters
```

Parameters

namelist-name
Namelist name. The namelist-name value should match the name of a namelist defined to the queue manager.

optional-parameters
Optional parameters for the ALTER NAMELIST command include:

DESCR(string)
Description.

NAMES(name, ...)
List of names. The names can be of any type, but must conform to the rules for naming WebSphere MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify NAMES(). The maximum number of names in the list is 256.

DEFINE NAMELIST

Purpose

Use DEFINE NAMELIST to define a new namelist to the queue manager.

Synonym

```
DEF NL
```

Syntax

```
DEFINE NAMELIST(namelist-name) optional-parameters
```

Parameters

namelist-name
Namelist name. The namelist-name value should be unique; it should not match a namelist name already defined to the queue manager.

optional-parameters
Optional parameters for the DEFINE NAMELIST command include:

DESCR(string)
Description.

NAMES(name, ...)
List of names. The names can be of any type, but must conform to the rules for naming WebSphere MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify NAMES(). The maximum number of names in the list is 256.

DELETE NAMELIST

Purpose

Use DELETE NAMELIST to delete a namelist definition.

Synonym

```
DELETE NL
```

Syntax

```
DELETE NAMELIST(namelist-name)
```

Parameters

namelist-name
Namelist name. The namelist-name value should match an existing namelist defined to the queue manager.

DISPLAY NAMELIST

Purpose

Use DISPLAY NAMELIST to display a namelist definition.

Synonym

```
DIS NL
```

Syntax

```
DISPLAY   NAMELIST(namelist-name)
WHERE (FilterCondition) requested-attributes
```

Parameters

namelist-name

Namelist name. The namelist-name value should match an existing namelist defined to the queue manager.

The namelist-name can be generic. A trailing asterisk (*) matches all namelists with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all namelists. Characters after the first asterisk, if present, are ignored.

requested-attributes

Attributes of the namelist that are to be displayed.

This can be:

ALL Displays all namelist attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

DESCR
Namelist description.

NAMCOUNT
Number of namelist names.

NAMES
Namelist names.

Examples

```
DISPLAY NAMELIST(*) WHERE(DESCR LK 'ABC DESCR*')
DISPLAY NAMELIST(*) WHERE(ALTDATE LK 2010-08*)  NAMCOUNT  ALTTIME
DISPLAY NAMELIST(*) WHERE(ALTTIME NL '07.13*')
DISPLAY NAMELIST(*) WHERE(NAMCOUNT GT 5) +
        DESCR ALTDATE ALTTIME NAMCOUNT
DISPLAY NAMELIST(ABC*) WHERE(NAMES EX MY.OBJ.NAME.EXCLUDE)
```

WebSphere MQ queue commands

The WebSphere MQ queue commands are:

- ALTER QALIAS
- ALTER QLOCAL
- ALTER QMODEL
- ALTER QREMOTE
- DEFINE QALIAS

- DEFINE QLOCAL
- DEFINE QMODEL
- DEFINE QREMOTE
- DELETE QALIAS
- DELETE QLOCAL
- DELETE QMODEL
- DELETE QREMOTE
- DISPLAY QALIAS
- DISPLAY QLOCAL
- DISPLAY QMODEL
- DISPLAY QREMOTE
- DISPLAY QSTATUS

Note: Commands that alter queue attributes require that the relevant queue is not currently in use. In addition, since the system command and reply queues are used by the MQPMQSC utility, these queues are always in use and cannot be altered using the MQPMQSC program.

ALTER QALIAS

Purpose

Use ALTER QALIAS to alter the parameters of an alias queue.

Synonym

```
ALT QA
```

Syntax

```
ALTER QALIAS(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify an existing alias queue name defined to the queue manager.

optional-parameters
optional-parameters for the ALTER QALIAS command include:

DESCR(string)
Alias queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

PUT(ENABLED/DISABLED)
Put uninhibit and inhibit.

TARGQ(string)
Target queue of alias.

ALTER QLOCAL

Purpose

Use ALTER QLOCAL to alter the parameters of a local queue.

Synonym

```
ALT QL
```

Syntax

```
ALTER QLOCAL(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify an existing local queue name defined to the queue manager.

optional-parameters
optional-parameters for the ALTER QLOCAL command include:

ACCTQ(QMGR/OFF/ON)
Queue accounting setting.

DESCR(string)
Local queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

MAXDEPTH(integer)
Maximum queue depth.

MAXMSGL(integer)
Maximum message length.

MAXQUSER(integer)
Maximum number of active opens.

MAXGLOCK(integer)
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK(integer)
Buffer size for applications to manage concurrent queue access.

MAXTRIGS(integer)
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PROPCTL(COMPAT/ALL/NONE/FORCE)
Property control attribute.

This parameter is applicable to Local, Alias, and Model queues.

This parameter is optional.

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. Permissible values are:
ALL To contain all the properties of the message, except those

contained in the message descriptor (or extension), select All. The All value enables applications that cannot be changed to access all the message properties from MQRFH2 headers.

COMPAT
If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise, all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

FORCE
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application.

PUT(ENABLED/DISABLED)
PUT uninhibit and inhibit.

QDEPTHHI(integer)
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO(integer)
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV(ENABLED/DISABLED)
Controls whether Queue Depth High events are generated.

QDPLOEV(ENABLED/DISABLED)
Controls whether Queue Depth Low events are generated.

QDPMAXEV(ENABLED/DISABLED)
Controls whether Queue Full events are generated.

QSVCIEV(HIGH/OK/NONE)
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT(integer)
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

REORG(ENABLED/DISABLED)
Automatic VSAM reorganization.

REORGTI(string)
Automatic VSAM reorganization start time, in HHMM format.

REORGINT(integer)
Automatic VSAM reorganization interval.

REORGCAT(string)
Automatic VSAM reorganization catalog. No longer used.

For the reorganization process, the VSAM catalog where the reorganization file is defined is now extracted from the system and so no longer needs to be specified in the queue definition. If specified it is treated as a comment.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN(string)
Channel name for MCA trigger process.

TRIGDATA(string)
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG(string)
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM(string)
Terminal identifier for trigger process.

TRIGTRAN(string)
Transaction identifier for trigger process.

TRIGTYPE(FIRST/EVERY)
Trigger type.

USAGE(NORMAL/XMIT)
Queue usage.

ALTER QMODEL

Purpose

Use ALTER QMODEL to alter the parameters of a model queue.

Synonym

```
ALT QM
```

Syntax

```
ALTER QMODEL(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify an existing model queue name defined to the queue manager.

optional-parameters
optional-parameters for the ALTER QMODEL command include:

ACCTQ(QMGR/OFF/ON)
Queue accounting setting.

DEFTYPE(TEMPDYN/PERMDYN)
Definition type.

DESCR(string)
Local queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

MAXDEPTH(integer)
Maximum queue depth.

MAXMSGL(integer)
Maximum message length.

MAXQUSER(integer)
Maximum number of active opens.

MAXGLOCK(integer)
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK(integer)
Buffer size for applications to manage concurrent queue access.

MAXTRIGS(integer)
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PUT(ENABLED/DISABLED)
PUT uninhibit and inhibit.

QDEPTHHI(integer)
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO(integer)
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV(ENABLED/DISABLED)
Controls whether Queue Depth High events are generated.

QDPLOEV(ENABLED/DISABLED)
Controls whether Queue Depth Low events are generated.

QDPMAXEV(ENABLED/DISABLED)
Controls whether Queue Full events are generated.

QSVCIEV(HIGH/OK/NONE)
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT(integer)
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN(string)
Channel name for MCA trigger process.

TRIGDATA(string)
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG(string)
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM(string)
Terminal identifier for trigger process.

TRIGTRAN(string)
Transaction identifier for trigger process.

TRIGTYPE(FIRST/EVERY)
Trigger type.

USAGE(NORMAL/XMIT)
Queue usage.

ALTER QREMOTE

Purpose

Use ALTER QREMOTE to alter the parameters of a remote queue.

Synonym

```
ALT QR
```

Syntax

```
ALTER QREMOTE(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify an existing remote queue name defined to the queue manager.

optional-parameters
optional-parameters for the ALTER QREMOTE command include:

PUT(ENABLED/DISABLED)
Put inhibit and uninhibit.

RNAME(string)
Remote queue name.

RQMNAME(string)
Remote queue manager name.

XMITQ(string)
Transmission queue name.

DEFINE QALIAS
Purpose

Use DEFINE QALIAS to define a new alias queue, and set its parameters.

Note: An alias queue provides a level of indirection to another queue. The queue to which the alias refers must be another local or remote queue, defined at this queue manager. It cannot be another alias queue.

Synonym

```
DEF QA
```

Syntax

```
DEFINE QALIAS(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify a unique queue name that is not already defined to the queue manager.

optional-parameters
optional-parameters for the DEFINE QALIAS command include:

DESCR(string)
Alias queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

PUT(ENABLED/DISABLED)
Put uninhibit and inhibit.

TARGQ(string)
Target queue of alias.

To define an ALIAS REPLY QUEUE specify:

```
DEFINE QALIAS(alias-reply-queue-name) +
  DESCR('description') +
  RNAME(replyto-q-name) +
  RQMNAME(reply-to-qmgr-name)
```

To define an ALIAS QUEUE MANAGER specify:

```
DEFINE QALIAS(alias-qmgr-name) +
  DESCR('description') +
```

```
RQMNAME(qmgr-name) +
TARGOQ(xmit-q-name)
```

DEFINE QLOCAL

Purpose

Use DEFINE QLOCAL to define a new local queue, and set its parameters.

Synonym

```
DEF QL
```

Syntax

```
DEFINE QLOCAL(q-name) CICSFILE(f-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify a unique queue name that is not already defined to the queue manager.

f-name
CICS file name for queue messages. The f-name value should specify a filename defined to the CICS region.

optional-parameters
optional-parameters for the DEFINE QLOCAL command include:

ACCTQ(QMGR/OFF/ON)
Queue accounting setting.

DESCR(string)
Local queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

MAXDEPTH(integer)
Maximum queue depth.

MAXMSGL(integer)
Maximum message length.

MAXQUSER(integer)
Maximum number of active opens.

MAXGLOCK(integer)
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK(integer)
Buffer size for applications to manage concurrent queue access.

MAXTRIGS(integer)
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PROPCTL(COMPAT/ALL/NONE/FORCE)
Property control attribute.

PUT(ENABLED/DISABLED)
Put uninhibit and inhibit.

QDEPTHHI(integer)
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO(integer)
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV(ENABLED/DISABLED)
Controls whether Queue Depth High events are generated.

QDPLOEV(ENABLED/DISABLED)
Controls whether Queue Depth Low events are generated.

QDPMAXEV(ENABLED/DISABLED)
Controls whether Queue Full events are generated.

QSVCIEV(HIGH/OK/NONE)
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT(integer)
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

REORG(ENABLED/DISABLED)
Automatic VSAM reorganization.

REORGTI(string)
Automatic VSAM reorganization start time, in HHMM format.

REORGINT(integer)
Automatic VSAM reorganization interval.

REORGCAT(string)
Automatic VSAM reorganization catalog.

The contents of this field are now treated as comments. For the reorganization process the VSAM catalog where the reorganization file is defined is now extracted from the system and so no longer needs to be specified in the queue definition.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN(string)
Channel name for MCA trigger process.

TRIGDATA(string)
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG(string)
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM(string)
Terminal identifier for trigger process.

TRIGTRAN(string)
Transaction identifier for trigger process.

TRIGTYPE(FIRST/EVERY)
Trigger type.

USAGE(NORMAL/XMIT)
Queue usage.

DEFINE QMODEL

Purpose

Use DEFINE QMODEL to define a new model queue, and set its parameters.

Synonym

```
DEF QM
```

Syntax

```
DEFINE QMODEL(q-name) CICSFILE(f-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify a unique queue name that is not already defined to the queue manager.

f-name
CICS file name for queue messages. The f-name value should specify a filename defined to the CICS region.

optional-parameters
optional-parameters for the DEFINE QMODEL command include:

ACCTQ(QMGR/OFF/ON)
Queue accounting setting.

DEFTYPE(TEMPDYN/PERMDYN)
Definition type.

DESCR(string)
Local queue description.

GET(ENABLED/DISABLED)
Get uninhibit and inhibit.

MAXDEPTH(integer)
Maximum queue depth.

MAXMSGL(integer)
Maximum message length.

MAXQUSER(integer)
Maximum number of active opens.

MAXGLOCK(integer)
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK(integer)
Buffer size for applications to manage concurrent queue access.

MAXTRIGS(integer)
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PROPCTL(COMPAT/ALL/FORCE/NONE)
Property control attribute.

PUT(ENABLED/DISABLED)
Put uninhibit and inhibit.

QDEPTHHI(integer)
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO(integer)
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV(ENABLED/DISABLED)
Controls whether Queue Depth High events are generated.

QDPLOEV(ENABLED/DISABLED)
Controls whether Queue Depth Low events are generated.

QDPMAXEV(ENABLED/DISABLED)
Controls whether Queue Full events are generated.

QSVCIEV(HIGH/OK/NONE)
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT(integer)
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN(string)
Channel name for MCA trigger process.

TRIGDATA(string)
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG(string)
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM(string)
Terminal identifier for trigger process.

TRIGTRAN(string)
Transaction identifier for trigger process.

TRIGTYPE(FIRST/EVERY)
Trigger type.

USAGE(NORMAL/XMIT)
Queue usage.

DEFINE QREMOTE

Purpose

Use DEFINE QREMOTE to define a new remote queue, and set its parameters.

Synonym

```
DEF QR
```

Syntax

```
DEFINE QREMOTE(q-name) optional-parameters
```

Parameters

q-name
Queue name. The q-name value should specify a unique queue name that is not already defined to the queue manager.

optional-parameters
optional-parameters for the DEFINE QREMOTE command include:

DESCR(string)
Remote queue description.

PUT(ENABLED/DISABLED)
Put uninhibit and inhibit.

RNAME(string)
Remote queue name.

RQMNAME(string)
Remote queue manager name.

XMITQ(string)
Transmission queue name.

DELETE QALIAS

Purpose

Use DELETE QALIAS to delete an alias queue definition.

Synonym

```
DELETE QA
```

Syntax

```
DELETE QALIAS(q-name)
```

Parameters

q-name
Queue name. The q-name value should specify an existing alias queue name defined to the queue manager.

DELETE QLOCAL

Purpose

Use DELETE QLOCAL to delete a local queue definition.

Synonym

```
DELETE QL
```

Syntax

```
DELETE QLOCAL(q-name) optional-parameter
```

Parameters

q-name
Queue name. The q-name value should specify an existing local queue name defined to the queue manager.

optional-parameter
The optional-parameters for the DELETE QLOCAL command is:

PURGE
Purge queue messages. If the queue contains messages and the PURGE parameter is not specified, the command will fail.

DELETE QMODEL

Purpose

Use DELETE QMODEL to delete a model queue definition.

Synonym

```
DELETE QM
```

Syntax

```
DELETE QMODEL(q-name)
```

Parameters

q-name
Queue name. The q-name value should specify an existing model queue name defined to the queue manager.

DELETE QREMOTE
Purpose

Use DELETE QREMOTE to delete a remote queue definition.

Synonym

```
DELETE QR
```

Syntax

```
DELETE QREMOTE(q-name)
```

Parameters

q-name
Queue name. The q-name value should specify an existing remote queue name defined to the queue manager.

DISPLAY QALIAS
Purpose

Use DISPLAY QALIAS to display the attributes of an alias queue.

Synonym

```
DIS QA
```

Syntax

```
DISPLAY   QALIAS(q-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

q-name
Queue name. The q-name value should specify an existing alias queue name defined to the queue manager.

The q-name can be generic. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues. Characters after the first asterisk, if present, are ignored.

requested-attributes
Attributes of the alias queue that are to be displayed. This can be:

ALL Displays all alias queue attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

DEFTYPE
Definition type.

DESCR
Alias queue description.

GET GET inhibit and uninhibit.

PUT PUT inhibit and uninhibit.

TARGQ(string)
Target queue of alias.

Examples

```
DISPLAY QALIAS(ABC*)  WHERE(DESCR LK TEST*)
DISPLAY QALIAS(ABC*)  WHERE(TARGQ GE ANYQ)
```

DISPLAY QLOCAL

Purpose

Use DISPLAY QLOCAL to display the attributes of a local queue.

Synonym

```
DIS QL
```

Syntax

```
DISPLAY   QLOCAL(q-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

q-name
Queue name. The q-name value should specify an existing local queue name defined to the queue manager.

The q-name can be generic. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues. Characters after the first asterisk, if present, are ignored.

requested-attributes
Attributes of the local queue that are to be displayed. This can be:

ACCTQ(QMGR/OFF/ON)
Queue accounting setting.

ALL Displays all local queue attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

CICSFILE
CSD file name for queue messages.

DEFPSIST
Default message persistence.

DEFTYPE
Definition type.

DESCR
Local queue description.

GET Get inhibit and uninhibit.

MAXDEPTH
Maximum queue depth.

MAXMSGL
Maximum message length.

MAXQUSER
Maximum number of active opens.

MAXGLOCK
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK
Buffer size for applications to manage concurrent queue access.

MAXTRIGS
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PROPCTL
Property control attribute.

PUT PUT inhibit and uninhibit.

QDEPTHHI
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV
Controls whether Queue Depth High events are generated.

QDPLOEV
Controls whether Queue Depth Low events are generated.

QDPMAXEV
Controls whether Queue Full events are generated.

QSVCIEV
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

REORG
Automatic VSAM reorganization.

REORGTI
Automatic VSAM reorganization start time, in HHMM format.

REORGINT
Automatic VSAM reorganization interval.

REORGCAT
Automatic VSAM reorganization catalog.

The contents of this field are now treated as comments and may not reflect the actual VSAM catalog containing the reorganization file. For the reorganization process, the VSAM catalog where the reorganization file is defined is now extracted from the system.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN
Channel name for MCA trigger process.

TRIGDATA
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM
Terminal identifier for trigger process.

TRIGTRAN
Transaction identifier for trigger process.

TRIGTYPE
Trigger type.

USAGE
Queue usage.

Examples

```
DISPLAY QLOCAL(ABC*)  WHERE(ACCTQ EQ QMGR)
DISPLAY QLOCAL(ABC*)  WHERE(ALTDATE GE 2010-08-23)
DISPLAY QLOCAL(ABC*)  WHERE(ALTTIME GT 05.00.00)
DISPLAY QLOCAL(ABC*)  WHERE(CICSFILE LK 'MQFI*')
DISPLAY QLOCAL(ABC*)  WHERE(DEFPSIST EQ YES)
DISPLAY QLOCAL(ABC*)  WHERE(DESCR LK 'QUEUE MODEL*')
DISPLAY QLOCAL(ABC*)  WHERE(DISTL EQ YES)
```

```
DISPLAY QLOCAL(*)     WHERE(GET EQ DISABLED)
DISPLAY QLOCAL(ABC*)  WHERE(MAXDEPTH GE 10000)
DISPLAY QLOCAL(*)     WHERE(MAXGLOCK GT 100)
DISPLAY QLOCAL(*)     WHERE(MAXLLOCK NE 80)
DISPLAY QLOCAL(ABC*)  WHERE(MAXQUSER LT 90)
DISPLAY QLOCAL(ABC*)  WHERE(MAXTRIGS EQ 5)
DISPLAY QLOCAL(*)     WHERE(MONQ NE OFF)
DISPLAY QLOCAL(ABC*)  WHERE(MSGDLVSQ EQ FIFO)
DISPLAY QLOCAL(ABC*)  WHERE(QDEPTHHI GE 95)
DISPLAY QLOCAL(ABC*)  WHERE(QSVCINT LE 100)
DISPLAY QLOCAL(ABC*)  WHERE(REORG EQ DISABLED)
DISPLAY QLOCAL(ABC*)  WHERE(REORGINT GT 60)
DISPLAY QLOCAL(ABC*)  WHERE(REORGTI LE 0400)
DISPLAY QLOCAL(ABC*)  WHERE(SHARE EQ YES)
DISPLAY QLOCAL(ABC*)  WHERE(STATQ EQ OFF)
DISPLAY QLOCAL(*)     WHERE(TRIGCHAN GT ABC.CHANNEL.NAME)
DISPLAY QLOCAL(*)     WHERE(TRIGDATA NE USER_DATA)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGGER EQ NO)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGPROG EQ MQPSEND)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGREST EQ NO)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGTERM LK A*)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGTRAN EQ TRNX)
DISPLAY QLOCAL(ABC*)  WHERE(TRIGTYPE EQ EVERY)
DISPLAY QLOCAL(ABC*)  WHERE(USAGE NE XMIT)
```

DISPLAY QMODEL

Purpose

Use DISPLAY QMODEL to display the attributes of a model queue.

Synonym

```
DIS QM
```

Syntax

```
DISPLAY   QMODEL(q-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

q-name

Queue name. The q-name value should specify an existing model queue name defined to the queue manager.

The q-name can be generic. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues. Characters after the first asterisk, if present, are ignored.

requested-attributes

Attributes of the model queue that are to be displayed. This can be:

ACCTQ(QMGR/OFF/ON)

Queue accounting setting.

ALL Displays all local queue attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE

Last modification date.

ALTTIME

Last modification time.

CICSFILE
CSD file name for queue messages.

DEFPSIST
Default message persistence.

DEFTYPE
Definition type.

DESCR
Local queue description.

GET Get inhibit and uninhibit.

MAXDEPTH
Maximum queue depth.

MAXMSGL
Maximum message length.

MAXQUSER
Maximum number of active opens.

MAXGLOCK
Buffer size for queue manager to manage concurrent queue access.

MAXLLOCK
Buffer size for applications to manage concurrent queue access.

MAXTRIGS
Maximum number of concurrent trigger instances.

MONQ(QMGR/OFF/LOW/MEDIUM/HIGH)
Queue monitoring setting.

NOSHARE
Non-shareable queue. This parameter is mutually exclusive to the SHARE parameter.

NOTRIGGER
No trigger on queue. This parameter is mutually exclusive to the TRIGGER parameter.

NOTRIGREST
No trigger restart allowed. This parameter is mutually exclusive to the TRIGREST parameter.

PROPCTL
Property control attribute.

PUT Put inhibit and uninhibit.

QDEPTHHI
The threshold against which the queue depth is compare to generate a Queue Depth High event.

QDEPTHLO
The threshold against which the queue depth is compare to generate a Queue Depth Low event.

QDPHIEV
Controls whether Queue Depth High events are generated.

QDPLOEV
Controls whether Queue Depth Low events are generated.

QDPMAXEV
Controls whether Queue Full events are generated.

QSVCIEV
Controls whether Service Interval High or Service Interval OK events are generated.

QSVCINT
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

SHARE
Shareable queue. This parameter is mutually exclusive to the NOSHARE parameter.

STATQ(QMGR/ON/OFF)
Queue statistics setting.

TRIGCHAN
Channel name for MCA trigger process.

TRIGDATA
User data passed to trigger instance.

TRIGGER
Trigger on queue. This parameter is mutually exclusive to the NOTRIGGER parameter.

TRIGPROG
Program name for trigger process.

TRIGREST
Trigger restart allowed. This parameter is mutually exclusive to the NOTRIGREST parameter.

TRIGTERM
Terminal identifier for trigger process.

TRIGTRAN
Transaction identifier for trigger process.

TRIGTYPE
Trigger type.

USAGE
Queue usage.

Examples

```
DISPLAY QMODEL(ABC*)  WHERE(ACCTQ EQ QMGR)
DISPLAY QMODEL(ABC*)  WHERE(ALTDATE GE 2010-08-23)
DISPLAY QMODEL(ABC*)  WHERE(ALTTIME GT 05.00.00)
DISPLAY QMODEL(ABC*)  WHERE(CICSFILE LK 'MQFI*')
DISPLAY QMODEL(ABC*)  WHERE(DEFTYPE EQ PERMDYN)
DISPLAY QMODEL(ABC*)  WHERE(DESCR LK 'QUEUE MODEL*')
DISPLAY QMODEL(*)     WHERE(GET EQ DISABLED)
DISPLAY QMODEL(ABC*)  WHERE(MAXDEPTH GE 10000)
DISPLAY QMODEL(*)     WHERE(MAXGLOCK GT 100)
DISPLAY QMODEL(*)     WHERE(MAXLLOCK NE 80)
DISPLAY QMODEL(ABC*)  WHERE(MAXQUSER LT 90)
DISPLAY QMODEL(ABC*)  WHERE(MAXTRIGS EQ 5)
DISPLAY QMODEL(*)     WHERE(MONQ NE OFF)
DISPLAY QMODEL(ABC*)  WHERE(MSGDLVSQ EQ FIFO)
DISPLAY QMODEL(ABC*)  WHERE(QDEPTHHI GE 95)
DISPLAY QMODEL(ABC*)  WHERE(QSVCINT LE 100)
DISPLAY QMODEL(ABC*)  WHERE(SHARE EQ YES)
```

```
DISPLAY QMODEL(ABC*)  WHERE(STATQ EQ OFF)
DISPLAY QMODEL(*)     WHERE(TRIGCHAN GT ABC.CHANNEL.NAME)
DISPLAY QMODEL(*)     WHERE(TRIGDATA NE USER_DATA)
DISPLAY QMODEL(ABC*)  WHERE(TRIGGER EQ NO)
DISPLAY QMODEL(ABC*)  WHERE(TRIGPROG EQ MQPSEND)
DISPLAY QMODEL(ABC*)  WHERE(TRIGREST EQ NO)
DISPLAY QMODEL(ABC*)  WHERE(TRIGTERM LK A*)
DISPLAY QMODEL(ABC*)  WHERE(TRIGTRAN EQ TRNX)
DISPLAY QMODEL(ABC*)  WHERE(USAGE NE XMIT)
```

DISPLAY QREMOTE

Purpose

Use DISPLAY QREMOTE to display the attributes of a remote queue.

Synonym

```
DIS QR
```

Syntax

```
DISPLAY   QREMOTE(q-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

q-name

Queue name. The q-name value should specify an existing remote queue name defined to the queue manager.

The q-name can be generic. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues. Characters after the first asterisk, if present, are ignored.

requested-attributes

Attributes of the remote queue that are to be displayed. This can be:

ALL Displays all remote queue attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

DESCR
Descriptive comment.

PUT Put inhibit and uninhibit.

RNAME
Remote queue name.

RQMNAME
Remote queue manager name.

XMITQ
Transmission queue name.

Examples

```
DISPLAY QREMOTE(ABC*)  WHERE(ALTDATE GE 2010-08-23)
DISPLAY QREMOTE(ABC*)  WHERE(ALTTIME GT 05.42.27)
DISPLAY QREMOTE(ABC*)  WHERE(DESCR GT 'QUEUE REMOTE 1')
```

```
DISPLAY QREMOTE(*)     WHERE(PUT EQ DISABLED)
DISPLAY QREMOTE(ABC*)  WHERE(RNAME LK  ANYQ*)
DISPLAY QREMOTE(ABC*)  WHERE(RQMNAME NL  'TEST*')
DISPLAY QREMOTE(ABC*)  WHERE(XMITQ NL  'TEST.XQ*')
```

DISPLAY QSTATUS

Purpose

Use the MQSC command DISPLAY QSTATUS to display the status of one or more queues.

Synonym

```
DIS QS
```

Syntax

```
DISPLAY   QSTATUS(generic-qname)
WHERE(FilterCondition) requested-attributes
```

Usage

You must specify the name of the queue for which you want to display status information. This can be a specific queue name or a generic queue name. By using a generic queue name you can display either:

- Status information for all queues, or
- Status information for one or more queues that match the specified name and other selection criteria.

You must also specify whether you want status information about:

- Queues.
- Handles that are accessing the queues.

Note: You cannot use the DISPLAY QSTATUS command to display the status of an alias queue or remote queue. If you specify the name of one of these types of queue, no data is returned. You can, however, specify the name of the local queue or transmission queue to which the alias queue or remote queue resolves.

Parameters

(generic-qname)

The name of the queue for which status information is to be displayed. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own matches all queues.

requested-attributes

ALL Display all the status information for each specified queue.

This is the default if you do not specify a generic name, and do not request any specific parameters.

MONITOR

Specify this to return the set of online monitoring parameters. These are LGETDATE, LGETTIME, LPUTDATE, LPUTTIME, MONQ, MSGAGE, and QTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

OPENTYPE
Restricts the queues selected to those that have handles with the specified type of access:
- ALL Selects queues that are open with any type of access. This is the default if the OPENTYPE parameter is not specified.
- INPUT Selects queues that are open for input only. This option does not select queues that are open for browse.
- OUTPUT Selects queues that are open only for output.

TYPE Specifies the type of status information required:
- QUEUE Status information relating to queues is displayed. This is the default if the TYPE parameter is not specified.
- HANDLE Status information relating to the handles that are accessing the queues is displayed.

Queue status:
For queue status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:
- Queue name.
- Type of information returned (TYPE parameter).
- Current queue depth (CURDEPTH parameter).

The following parameters can be specified for TYPE(QUEUE) to request additional information for each queue. If a parameter is specified that is not relevant for the queue, operating environment, or type of status information requested, that parameter is ignored.

CURDEPTH The current depth of the queue, that is, the
number of messages on the queue. This includes both committed messages and uncommitted messages.

IPPROCS The number of handles that are currently open for
input for the queue (either input-shared or input-exclusive). This does not include handles that are open for browse.

LGETDATE The date on which the last message was retrieved
from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get date is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank.

This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LGETTIME
The time at which the last message was retrieved from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get time is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank.

This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTDATE
The date on which the last message was put to the queue since the

queue manager started. When no put date is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank.

This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTTIME
The time at which the last message was put to the queue since the queue manager started. When no put time is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank.

This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

MONQ
Current level of monitoring data collection for the queue. This parameter is also displayed when you specify the MONITOR parameter.

MSGAGE
Age, in seconds, of the oldest message on the queue. The maximum displayable value is 999 999 999; if the age exceeds this value, 999 999 999 is displayed.

This parameter is also displayed when you specify the MONITOR parameter. A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

OPPROCS
This is the number of handles that are currently open for output for the queue.

QTIME
Interval, in microseconds, between messages being put on the queue and then being destructively read. The maximum displayable value is 999 999 999; if the interval exceeds this value, 999 999 999 is displayed.

The interval is measured from the time that the message is placed on the queue until it is retrieved by an application and, therefore, includes any interval caused by a delay in committing by the putting application.

Two values are displayed:
- A value based on recent activity over a short period of time.
- A value based on activity over a longer period of time.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

UNCOM
Indicates whether there are any uncommitted changes (puts and gets) pending for the queue. The value displayed is one of the following:
YES There are one or more uncommitted changes pending.
NO There are no uncommitted changes pending.
N An integer value indicating how many uncommitted changes are pending.

Handle status:
For handle status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:
- Queue name
- Type of information returned (TYPE parameter).
- User identifier (USERID parameter).
- Application tag (APPLTAG parameter).
- Application type (APPLTYPE parameter).
- Whether handle is providing input access (INPUT parameter).
- Whether handle is providing output access (OUTPUT parameter).
- Whether handle is providing browse access (BROWSE parameter).
- Whether handle is providing inquire access (INQUIRE parameter).
- Whether handle is providing set access (SET parameter).

The following parameters can be specified for TYPE(HANDLE) to request additional information for each queue. If a parameter that is not relevant is specified for the queue, operating environment, or type of status information requested, that parameter is ignored.

APPLTAG
A string containing the tag of the application connected to the queue manager. It is one of the following:
- CICS APPLID

APPLTYPE
A string indicating the type of the application that is connected to the queue manager. It is one of the following:
BATCH
Application using a batch connection
CICS CICS transaction

BROWSE
Indicates whether the handle is providing browse access to the queue. The value is one of the following:
YES The handle is providing browse access.
NO The handle is not providing browse access.

CHANNEL
The name of the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

CONNAME
The connection name associated with the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

This parameter is returned only when the handle belongs to the channel initiator.

HSTATE
Whether an API call is in progress. Possible values are:
ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.
INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

INPUT
Indicates whether the handle is providing input access to the queue. The value is one of the following:
SHARED
The handle is providing shared-input access.
EXCL The handle is providing exclusive-input access.
NO The handle is not providing input access.

INQUIRE
Indicates whether the handle is providing inquire access to the queue. The value is one of the following:
YES The handle is providing inquire access.
NO The handle is not providing inquire access.

OUTPUT
Indicates whether the handle is providing output access to the queue. The value is one of the following:
YES The handle is providing output access.
NO The handle is not providing output access.

SET Indicates whether the handle is providing set access to the queue. The value is one of the following:
YES The handle is providing set access.
NO The handle is not providing set access.

TASKNO
A 7-digit CICS task number is returned as a 7-byte string. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

TRANSID
A 4-character CICS transaction identifier. This parameter is returned only when the APPLTYPE parameter has the value CICS.

URTYPE
The type of unit of recovery as seen by the queue manager. It is one of the following:

- CICS

USERID
The user identifier associated with the handle. This parameter is not returned when APPLTYPE has the value SYSTEM.

Examples

```
DISPLAY QSTATUS(ABC*) TYPE(QUEUE)  WHERE(CURDEPTH LT 3)
DISPLAY QSTATUS(*)    WHERE(IPPROCS GT 1)
DISPLAY QSTATUS(XY*)  WHERE(LGETDATE GE 2010-09-12)
DISPLAY QSTATUS(ABC*) WHERE(LGETTIME LT 17.00.00)
DISPLAY QSTATUS(ABC*) WHERE(LPUTDATE GT 2010-09-12)
DISPLAY QSTATUS(ABC*) WHERE(LPUTTIME LK 17.09*)
DISPLAY QSTATUS(ABC*) WHERE(MONQ EQ OFF)
DISPLAY QSTATUS(ABC*) WHERE(MSGAGE LT 3000)
DISPLAY QSTATUS(ABC*) WHERE(OPPROCS GT 1)
DISPLAY QSTATUS(*)    WHERE(UNCOM EQ YES)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE)WHERE(APPLTAG EQ TSMQ300)
DISPLAY QSTATUS(*)    TYPE(HANDLE) WHERE(APPLTYPE EQ BATCH)
DISPLAY QSTATUS(*)    TYPE(HANDLE) OPENTYPE(ALL) +
                      WHERE(BROWSE EQ YES)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) +
                 WHERE(CHANNEL LK ABC.SENDER*)
DISPLAY QSTATUS(*)    TYPE(HANDLE) +
                      WHERE(CONNAME EQ '1.123.345.1(1418)')
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) OPENTYPE(INPUT) +
                                   WHERE(HSTATE EQ ACTIVE)
DISPLAY QSTATUS(*)    TYPE(HANDLE) WHERE(INPUT EQ EXCL)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) WHERE(INQUIRE EQ YES)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) WHERE(OUTPUT EQ NO)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) OPENTYPE(OUTPUT)+
                                   WHERE(SET EQ NO)
DISPLAY QSTATUS(ABC*) TYPE(HANDLE) WHERE(TASKNO LT 0002638)
DISPLAY QSTATUS(*)    TYPE(HANDLE) WHERE(TRANSID LK QED*)
DISPLAY QSTATUS(*)    TYPE(HANDLE) WHERE(URTYPE EQ CICS)
DISPLAY QSTATUS(*)    TYPE(HANDLE) WHERE(USERID NL IBM*)
```

WebSphere MQ queue manager commands

The WebSphere MQ queue manager commands are:

- ALTER QMGR
- DISPLAY QMGR
- PING QMGR
- START LISTENER

ALTER QMGR

Purpose

Use ALTER QMGR to alter the queue manager parameters for the local queue manager.

Synonym

```
ALT QMGR
```

Syntax

```
ALTER QMGR qmgr-attrs
```

Parameters

qmgr-attrs

Queue manager attributes for the ALTER QMGR command include the following:

ACCTCONO(ENABLED/DISABLED)

Specifies whether applications can override the settings of the ACCTQ and ACCTMQI queue manager parameters:

DISABLED
Applications cannot override the settings of the ACCTQ and ACCTMQI parameters. This is the queue manager's initial default value.

ENABLED
Applications can override the settings of the ACCTQ and ACCTMQI parameters by using the options field of the MQCNO structure of the MQCONNX API call.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

ACCTINT(integer)
The time interval, in seconds, at which intermediate accounting records are written.

Specify a value in the range 1 through 604 800.

The queue manager's initial default value is 1800.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

ACCTMQI(ON/OFF)
Specifies whether accounting information for MQI data is to be collected:

OFF MQI accounting data collection is disabled. This is the queue manager's initial default value.

ON MQI accounting data collection is enabled.If queue manager attribute ACCTCONO is set to ENABLED, the value of this parameter can be overridden using the options field of the MQCNO structure.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

ACCTQ(ON/OFF/NONE)
Specifies whether accounting data is to be collected for all queues:

OFF Accounting data collection is disabled for all queues which specify QMGR as the value for their ACCTQ parameter. This is the queue manager's initial default value.

ON Accounting data collection is enabled for all queues which specify QMGR as the value of their ACCTQ parameter.

NONE
Accounting data collection for all queues is disabled regardless of the value of the queue's ACCTQ parameter.

Changes to this parameter are effective only for connections to the queue manager occurring after the change to the parameter.

ADOPTCHK(NONE/NETADDR)
Adopt MCA check.

ADOPTMCA(NO/RCVR)
Adopt MCA.

AUTHOREV(ENABLED/DISABLED)
Whether authorization (Not Authorized) events are generated.

BATCHID(string)
Batch interface identifier.

BIAUTO
Automatic activation of the batch interface. This parameter is mutually exclusive of the NOBIAUTO parameter.

CCSID(integer)
Coded character set identifier.

CHAD(ENABLED/DISABLED)
Whether receiver and server-connection channels can be defined automatically.

CHADEV(ENABLED/DISABLED)
Whether channel auto-definition events are generated.

CHADEXIT(string)
Auto-definition exit name. If this name is nonblank, the exit is called when an inbound request for an undefined receiver or server-connection channel is received.

CHLEV(ENABLED/DISABLED)
Whether channel events are generated.

CMDEV(string)
Specifies whether command events are generated:

DISABLED
Command events are not generated. This is the queue manager's initial default value.

ENABLED
Command events are generated for all successful commands.

NODISPLAY
Command events are generated for all successful commands other than DISPLAY commands.

COMMANDQ(string)
System command input queue.

CONFIGEV(string)
Whether configuration events are generated:

ENABLED
Configuration events are generated.

DISABLED
Configuration events are not generated. This is the queue manager's initial default value.

CSAUTO
Automatic activation of the PCF command server. This parameter is mutually exclusive of the NOCSAUTO parameter.

CSCNVRT
Data conversion by command server of PCF messages. This parameter is mutually exclusive of the NOCSCNVRT parameter.

CSDLQ
Dead letter queue store by command server of undeliverable PCF reply messages. This parameter is mutually exclusive of the NOCSDLQ parameter.

DEADQ(string)
Dead letter queue name.

DESCR(string)
Queue manager description.

INHIBTEV(ENABLED/DISABLED)
Whether inhibit (Inhibit Get and Inhibit Put) events are generated.

LISTPORT(integer)
TCP/IP listener port number.

LOCALEV(ENABLED/DISABLED)
Whether local error events are generated.

LOGQ(string)
System log queue name.

MAXCLNTS(integer)
Maximum number of concurrent client connections.

MAXRTASK(integer)
Maximum number of tasks for dual queue recovery.

MAXDEPTH(integer)
Maximum queue depth for local queues.

MAXGLOCK(integer)
Buffer size for queue manager to manage concurrent queue access.

MAXHANDS(integer)
Maximum number of connections to queue manager.

MAXLLOCK(integer)
Buffer size for applications to manage concurrent queue access.

MAXMSGL(integer)
Maximum length of messages for local queues.

MAXPROPL(integer)
The maximum length of property data in bytes that can be associated with a message.

MAXQOPEN(integer)
Maximum number of concurrently open queues.

MAXQUSER(integer)
Maximum number of active opens to any particular queue.

MONCHL(NONE/OFF/LOW/MEDIUM/HIGH)
Controls the collection of online monitoring data for channels:

OFF Online monitoring data collection is turned off for channels specifying a value of QMGR in their MONCHL parameter. This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for channels regardless of the setting of their MONCHL parameter.

LOW Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their MONCHL parameter.

MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGR in their MONCHL parameter.

HIGH Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their MONCHL parameter.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

MONINTVL(integer)
Queue manager housekeeping process interval.

MONITORQ(string)
MQI diagnostic queue name.

MONQ(NONE/OFF/LOW/MEDIUM/HIGH)
Controls the collection of online monitoring data for queues:

OFF Online monitoring data collection is turned off for queues specifying a value of QMGR in their MONQ parameter. This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for queues regardless of the setting of their MONQ parameter.

LOW Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

MEDIUM
Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

HIGH Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

Note that, in contrast to MONCHL, there is no distinction between the values LOW, MEDIUM and HIGH. These values all turn data collection on, but do not affect the rate of collection.

Changes to this parameter are effective only for queues opened after the parameter has been changed.

NOBIAUTO
Non-automatic activation of the batch interface. This parameter is mutually exclusive of the BIAUTO parameter.

NOCSAUTO
Non-automatic activation of the PCF command server. This parameter is mutually exclusive of the CSAUTO parameter.

NOCSCNVRT
No data conversion by command server of PCF messages. This parameter is mutually exclusive of the CSCNVRT parameter.

NOCSDLQ
No dead letter queue store by command server of undeliverable PCF reply messages. This parameter is mutually exclusive of the CSDLQ parameter.

OPTCCOMM(ENABLED/DISABLED/REPLY)
Whether communication messages are sent to the console.

OPTCCRIT(ENABLED/DISABLED/REPLY)
Whether critical messages are sent to the console.

OPTCERR(ENABLED/DISABLED/REPLY)
Whether error messages are sent to the console.

OPTCINFO(ENABLED/DISABLED)
Whether information messages are sent to the console.

OPTCORG(ENABLED/DISABLED/REPLY)
Whether reorganization messages are sent to the console.

OPTCSYS(ENABLED/DISABLED/REPLY)
Whether general system messages are sent to the console.

OPTCWARN(ENABLED/DISABLED)
Whether warning messages are sent to the console.

OPTECSMT(ENABLED/DISABLED)
Whether messages are written to the CICS CSMT when then system log queue is unavailable.

OPTIDUMP(ENABLED/DISABLED)
Whether the queue manager generates an internal dump following an application abnormal termination.

OPTLCOMM(ENABLED/DISABLED)
Whether communication messages are sent to the system log.

OPTLCRIT(ENABLED/DISABLED)
Whether critical messages are sent to the system log.

OPTLERR(ENABLED/DISABLED)
Whether error messages are sent to the system log.

OPTLINFO(ENABLED/DISABLED)
Whether information messages are sent to the system log.

OPTLORG(ENABLED/DISABLED)
Whether reorganization messages are sent to the system log.

OPTLSYS(ENABLED/DISABLED)
Whether general system messages are sent to the system log.

OPTLWARN(ENABLED/DISABLED)
Whether warning messages are sent to the system log.

OPTTCOMM(ENABLED/DISABLED)
Whether the queue manager traces communication related events.

OPTTCONV(ENABLED/DISABLED)
Whether the queue manager traces data conversion related events.

OPTTMQI(ENABLED/DISABLED)
Whether the queue manager traces MQI call related events.

OPTTORG(ENABLED/DISABLED)
Whether the queue manager traces reorganization related events.

OPTTSYS(ENABLED/DISABLED)
Whether the queue manager traces general system related events.

PERFMEV(ENABLED/DISABLED)
Whether performance-related events are generated.

REMOTEEV(ENABLED/DISABLED)
Whether remote error events are generated.

REPLYQ(string)
System command reply queue name for MQSC processing.

SSLEV(string)
Whether SSL events are generated.
DISABLED
SSL events are not generated. This is the queue manager's initial default value.
ENABLED
All SSL events are generated.

SSLKEYL(string)
SSL key library name.

SSLKEYM(string)
SSL key library member name.

SSLRKEYC(integer)
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.

This value is used only by SSL channels which initiate communication from the queue manager (for example, the sender channel in a sender and receiver channel pairing).

The count of bytes until the next secret key renegotiation is reset after each successful renegotiation.

Specify a value in the range zero through 999 999 999. A value of zero (the queue manager's initial default value) means that the secret key is never renegotiated. If you specify an SSL/TLS secret key reset count between 1 byte and 32Kb, SSL/TLS channels will use a secret key reset count of 32Kb. This is to avoid the cost of excessive key resets which would occur for small SSL/TLS secret key reset values.

STATCHL(NONE/OFF/LOW/MEDIUM/HIGH)
Whether statistics data is to be collected for channels:
NONE
Statistics data collection is turned off for channels regardless of the setting of their STATCHL parameter. This is the queue manager's initial default value.
. OFF Statistics data collection is turned off for channels specifying a value of QMGR in their STATCHL parameter.
. LOW Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.
MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.
. HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.

Note: A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

STATINT(integer)
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue. Specify a value in the range 1 through 604 800. The queue manager's initial default value is 1800.

Note: Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

STATMQI(ON/OFF)
Whether statistics monitoring data is to be collected for the queue manager:

OFF Data collection for MQI statistics is disabled. This is the queue manager's initial default value.

ON Data collection for MQI statistics is enabled.

Note: Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

STATQ(NONE/ON/OFF)
Whether statistics data is to be collected for queues:

NONE
Statistics data collection is turned off for queues regardless of the setting of their STATQ parameter. This is the queue manager's initial default value.

OFF Statistics data collection is turned off for queues specifying a value of QMGR in their STATQ parameter.

ON Statistics data collection is turned on for queues specifying a value of QMGR in their STATQ parameter.

Note: Changes to this parameter take immediate effect on the collection of statistics data for the affected queues.

STRSTPEV(ENABLED/DISABLED)
Whether start and stop events are generated.

DISPLAY QMGR

Purpose

Use DISPLAY QMGR to display the attributes of the queue manager.

Synonym

```
DIS QMGR
```

Syntax

```
DISPLAY QMGR requested-attributes
```

Parameters

requested-attributes
Attributes of the queue manager that are to be displayed. This can be:

ACCTCONO
Specifies whether applications can override the settings of the ACCTQ and ACCTMQI queue manager parameters.

ACCTINT
The time interval, in seconds, at which intermediate accounting records are written.

ACCTMQI
Specifies whether accounting information for MQI data is to be collected.

ACCTQ
Specifies whether accounting data is to be collected for all queues.

ADOPTCHK
Adopt MCA check.

ADOPTMCA
Adopt MCA.

ALL Displays all queue manager attributes. If you do not use ALL, you can use any combination of the other keywords.

ALTDATE
Last modification date.

ALTTIME
Last modification time.

AUTHOREV
Whether authorization (Not Authorized) events are generated.

BATCHID
Batch interface identifier.

BIAUTO
Automatic activation of the batch interface. This parameter is mutually exclusive of the NOBIAUTO parameter.

CCSID
Coded character set identifier.

CHAD
Whether auto-definition of receiver and server-connection channels is enabled.

CHADEV
Whether auto-definition events are enabled.

CHADEXIT
The name of the channel auto-definition exit.

CHLAUTH
Whether channel authentication records are checked.

CMDEV
Specifies whether command events are generated.

COMMANDQ
System command input queue.

CONFIGEV
Whether configuration events are generated.

CSAUTO
Automatic activation of the PCF command server. This parameter is mutually exclusive of the NOCSAUTO parameter.

CSCNVRT
Data conversion by command server of PCF messages. This parameter is mutually exclusive of the NOCSCNVRT parameter.

CSDLQ
Dead letter queue store by command server of undeliverable PCF reply messages. This parameter is mutually exclusive of the NOCSDLQ parameter.

CHLEV
Whether channel events are generated.

DEADQ
Dead letter queue name.

DESCR
Queue manager description.

INHIBTEV
Whether inhibit (Inhibit Get and Inhibit Put) events are generated.

LISTPORT
TCP/IP listener port number.

LOCALEV
Whether local error events are generated.

LOGQ
System log queue name.

MAXCLNTS
Maximum number of concurrent client connections.

MAXDEPTH
Maximum queue depth for local queues.

MAXGLOCK
Buffer size for queue manager to manage concurrent queue access.

MAXHANDS
Maximum number of connections to queue manager.

MAXLLOCK
Buffer size for applications to manage concurrent queue access.

MAXMSGL
Maximum length of messages for local queues.

MAXPROPL(integer)
The maximum length of property data in bytes that can be associated with a message.

MAXQOPEN
Maximum number of concurrently open queues.

MAXQUSER
Maximum number of active opens to any particular queue.

MAXRTASK
Maximum number of tasks for dual queue recovery.

MONCHL
Controls the collection of online monitoring data for channels.

MONINTVL
Queue manager housekeeping process interval.

MONITORQ
MQI diagnostic queue name.

MONQ
Controls the collection of online monitoring data for queues.

NOBIAUTO
Non-automatic activation of the batch interface. This parameter is mutually exclusive of the BIAUTO parameter.

NOCSAUTO
Non-automatic activation of the PCF command server. This parameter is mutually exclusive of the CSAUTO parameter.

NOCSCNVRT
No data conversion by command server of PCF messages. This parameter is mutually exclusive of the CSCNVRT parameter.

NOCSDLQ
No dead letter queue store by command server of undeliverable PCF reply messages. This parameter is mutually exclusive of the CSDLQ parameter.

OPTCCOMM
Whether communication messages are sent to the console.

OPTCCRIT
Whether critical messages are sent to the console.

OPTCERR
Whether error messages are sent to the console.

OPTCINFO
Whether information messages are sent to the console.

OPTCORG
Whether reorganization messages are sent to the console.

OPTCSYS
Whether general system messages are sent to the console.

OPTCWARN
Whether warning messages are sent to the console.

OPTECSMT
Whether messages are written to the CICS CSMT when then system log queue is unavailable.

OPTIDUMP
Whether the queue manager generates an internal dump following an application abnormal termination.

OPTLCOMM
Whether communication messages are sent to the system log.

OPTLCRIT
Whether critical messages are sent to the system log.

OPTLERR
Whether error messages are sent to the system log.

OPTLINFO
Whether information messages are sent to the system log.

OPTLORG
Whether reorganization messages are sent to the system log.

OPTLSYS
Whether general system messages are sent to the system log.

OPTLWARN
Whether warning messages are sent to the system log.

OPTTCOMM
Whether the queue manager traces communication related events.

OPTTCONV
Whether the queue manager traces data conversion related events.

OPTTMQI
Whether the queue manager traces MQI call related events.

OPTTORG
Whether the queue manager traces reorganization related events.

OPTTSYS
Whether the queue manager traces general system related events.

PERFMEV
Whether performance-related events are generated.

REMOTEEV
Whether remote error events are generated.

REPLYQ
System command reply queue name for MQSC processing.

SSLEV
Whether SSL events are generated.

SSLKEYL
SSL key library name.

SSLKEYM
SSL key library member name.

SSLRKEYC
The number of bytes to be sent and received within an SSL conversation before the secret key is renegotiated.

STATCHL
Whether statistics data is to be collected for channels.

STATINT
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue.

STATQ
Whether statistics data is to be collected for queues.

STRSTPEV
Whether start and stop events are generated.

PING QMGR
Purpose

Use PING QMGR to test whether the queue manager is responsive to commands.

Synonym

PING QMGR

Syntax

PING QMGR

WebSphere MQ Service

Service objects can be created, modified, deleted and displayed using MQSC commands. The following MQSC commands are supported:

ALTER SERVICE
DEFINE SERVICE
DELETE SERVICE
DISPLAY SERVICE
DISPLAY LSSTATUS
START SERVICE
STOP SERVICE

ALTER SERVICE

Purpose

Alter the parameters of an existing WebSphere MQ service definition. The service must be in STOPPED state.

Synonym

ALT SERVICE

Syntax

ALTER SERVICE(service-name) optional-parameters

Parameters

service-name
Name of the WebSphere MQ service definition to alter.

Optional-parameters
For details of optional parameters for the ALTER SERVICE command, see "DEFINE SERVICE."

DEFINE SERVICE

Purpose

Define a new WebSphere MQ service definition, and set its parameters.

Synonym

DEF SERVICE

Syntax

DEFINE SERVICE(service-name) optional-parameters

Parameters

service-name
Name of the WebSphere MQ service definition (see Rules for naming WebSphere MQ objects). This is required.

The name must not be the same as any other service definition currently defined on this queue manager.

Optional-parameters

Optional parameters for the DEFINE SERVICE command include:

CONTROL(string)

Specifies how the service is to be started and stopped:

MANUAL

The service is not to be started automatically or stopped automatically. It is to be 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.

DESCR(string)

Plain-text comment. It provides descriptive information about the service when an operator issues the DISPLAY SERVICE command (see DISPLAY SERVICE). It should contain only displayable characters. The maximum length is 64 characters.

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

SERVTYPE

Specifies the mode in which the service is to run:

COMMAND

A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

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 DISPLAY SVSTATUS command.

STARTARG(string)

Specifies the data to be passed in COMMAREA when starting the transaction specified for STARTCMD.

STARTCMD(string)

Specifies the CICS transaction code to be started to start service.

STOPARG(string)

Specifies the data to be passed in COMMAREA when starting the transaction specified for STOPCMD.

STOPCMD(string)

Specifies the CICS transaction code to be started to stop service.

DELETE SERVICE

Purpose

Delete a service definition. The service must be in STOPPED state.

Synonym

```
DEL SERVICE
```

Syntax

```
DELETE SERVICE(service-name)
```

Parameters

service-name
Name of the WebSphere MQ service definition to delete.

DISPLAY SERVICE

Purpose

Display information about a service. The values displayed describe the current definition of the service.

Synonym

```
DIS SERVICE
```

Syntax

```
DISPLAY   SERVICE(generic-service-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

generic-service-name
The name of the service definition for which information is to be displayed. A single asterisk (*) specifies that information for all service identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

Requested-attributes

ALL Specify this to display all the service information for each specified service. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed. This is the default if you do not specify a generic identifier, and do not request any specific parameters.

ALTDATE
The date on which the definition was last altered, in the form *yyyy-mm-dd*.

ALTTIME
The time at which the definition was last altered, in the form *hh.mm.ss*.

CONTROL
How the service is to be started and stopped:
MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service 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.

DESCR
Descriptive comment.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

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 DISPLAY SVSTATUS command.

STARTARG
The data to be passed in COMMAREA when starting the transaction specified for STARTCMD.

STARTCMD
Specifies the CICS transaction code to be started to start the service.

STOPARG
The data to be passed in COMMAREA when starting the transaction specified for STOPCMD.

STOPCMD
Specifies the CICS transaction code to be started to stop the service.

Examples

```
DISPLAY SERVICE(*)    WHERE(DESCR NL 'TEST*')
DISPLAY SERVICE(ABC*) WHERE(ALTDATE LK 2010-09*)
DISPLAY SERVICE(ABC*) WHERE(ALTTIME EQ 05.36.02)
DISPLAY SERVICE(ABC*) WHERE(STARTCMD NL MQ*)
DISPLAY SERVICE(ABC*) WHERE(STARTARG GT ' ')
DISPLAY SERVICE(ABC*) WHERE(STOPCMD EQ MQBI)
DISPLAY SERVICE(ABC*) WHERE(STOPARG EQ 'X')
DISPLAY SERVICE(ABC*) WHERE(CONTROL EQ QMGR)
DISPLAY SERVICE(ABC*) WHERE(SERVTYPE EQ SERVER)
```

DISPLAY LSSTATUS

Purpose

Display status information for one or more services. Only service with status of RUNNING are returned.

Synonym

```
DIS SVSTATUS
```

Syntax

```
DISPLAY   SVSTATUS(generic-service-name)
WHERE(FilterCondition) requested-attributes
```

Parameters

generic-service-name

The name of the service definition for which information is to be displayed. A single asterisk (*) specifies that information for all service identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

Requested-attributes

ALL Display all the status information for each specified service. This is the default if you do not specify a generic name, and do not request any specific parameters.

CONTROL

How the service is to be started and stopped:

MANUAL

The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR

The service 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.

DESCR

Descriptive comment.

PID The CICS task number associated with the service.

STARTARG

The data to be passed in COMMAREA when starting the transaction specified for STARTCMD.

STARTCMD

Specifies the CICS transaction code to be started.

STARTDA

The date on which the service was started.

STARTTI

The time at which the service was started.

STATUS

The current status of the process:

RUNNING

The service is running.

STOPPED

The service is stopping.

STOPARG

The data to be passed in COMMAREA when starting the transaction specified for STOPCMD.

STOPCMD
Specifies the CICS transaction code to be started when the service is stopped.

Examples

```
DISPLAY SVSTATUS(*) WHERE(DESCR LT 'INTERFACE TO BATCH JOBS')
DISPLAY SVSTATUS(*) WHERE(STARTDA EQ 2010-09-12)
DISPLAY SVSTATUS(*) WHERE(STARTTI GE 17.00.00)
DISPLAY SVSTATUS(*) WHERE(STARTCMD LK MQ*)
DISPLAY SVSTATUS(*) WHERE(STARTARG EQ ' ')
DISPLAY SVSTATUS(*) WHERE(STOPCMD GE MQBI)
DISPLAY SVSTATUS(*) WHERE(STOPARG NE X)
DISPLAY SVSTATUS(ABC*) WHERE(CONTROL EQ QMGR)
DISPLAY SVSTATUS(ABC*) WHERE(SERVTYPE EQ COMMAND)
DISPLAY SVSTATUS(*)    WHERE(STATUS EQ RUNNING)
```

START SERVICE
Purpose

Start a service.

Synonym

```
STA SERVICE
```

Syntax

```
START SERVICE(service-name)
```

Parameters

service-name
Name of the service to be started.

STOP SERVICE
Purpose

Stop a service.

Synonym

```
STOP SERVICE
```

Syntax

```
STOP SERVICE(service-name)
```

Parameters

service-name
Name of the service to be stopped.

WebSphere MQ Subscription

Subscription objects can be created, modified, deleted and displayed using MQSC commands. The following MQSC commands are supported:

- ALTER SUB
- DEFINE SUB
- DELETE SUB
- DISPLAY SUB
- DISPLAY SBSTATUS

ALTER SUB

Purpose

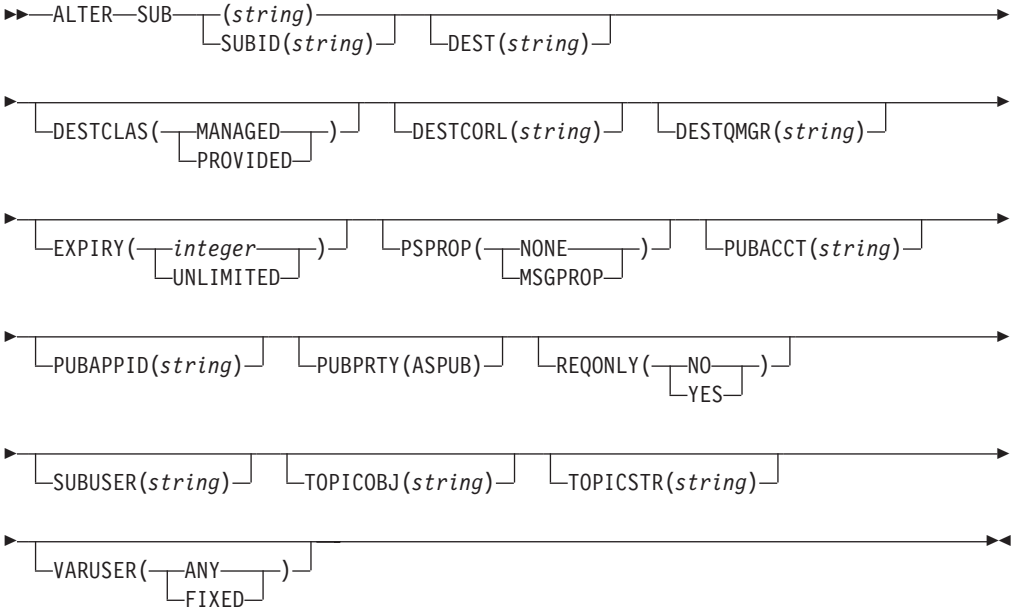
Use the MQSC command ALTER SUB to alter the characteristics of an existing subscription.

Parameters not specified in the ALTER SUB command result in the existing values for those parameters being left unchanged.

Synonym

ALT SUB

Syntax



Usage notes

The following are valid forms of the command:

```
ALT SUB('xyz')
ALT SUB SUBID(123)
ALT SUB(XYZ) SUBID(123)
```

Although permitted on the command, you cannot alter the following fields using ALTER SUB:

```
TOPICOBJ
TOPICSTRDESTCLAS
```

At the time the ALT SUB command processes, no check is performed that the named DEST or DESTQMGR exists. These names are used at publishing time as the ObjectName and ObjectQMgrName for an MQOPEN call. These names are resolved according to the WebSphere MQ name resolution rules.

Parameters

string A mandatory parameter. Specifies the unique name for this subscription, see SUBNAME property. If you wish to maintain the case of the name then

enclose in apostrophes. If you specify SUB(Abc) then the subname is ABC. You need to specify SUB('Abc') to maintain the lowercase characters.

DEST(*string***)**
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS
System managed destination.
PROVIDED
The destination is a queue.
MANAGED
The destination is managed.

DESTCORL(*string***)**
The CorrelId used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY
The time to expiry of the subscription object from the creation date and time.
integer The time to expiry, in tenths of a second, from the creation date and time.
UNLIMITED
There is no expiry time.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.
NONE
Do not add publish subscribe properties to the message.
MSGPROP
Publish subscribe properties are added as message properties.

PUBACCT(*string***)**
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the AccountingToken field of the MQMD.

PUBAPPID(*string***)**
Identity data passed by the subscriber, for propagation into messages published to this subscription in the ApplIdentityData field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB
Priority of the message sent to this subscription is taken from the priority supplied in the published message.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.
NO All publications on the topic are delivered to this subscription.
YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SUBUSER(*string***)**
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription.

TOPICSTR(*string***)**
Specifies a fully qualified topic name, or a topic set using wildcard character for the subscription.

TOPICOBJ(*string***)**
The name of a topic object used by this subscription.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED
Takeover by another USERID is not permitted.

DEFINE SUB

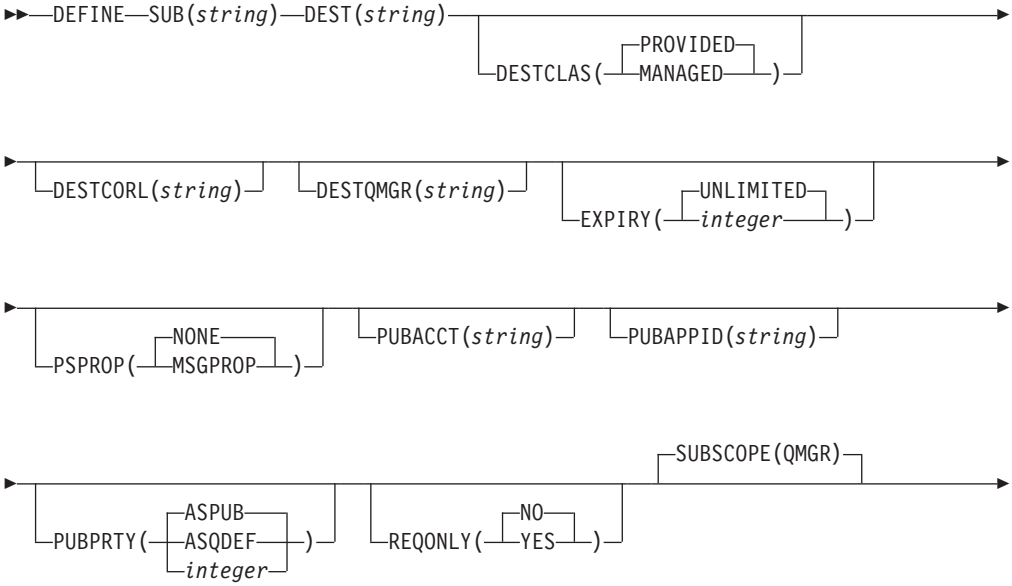
Purpose

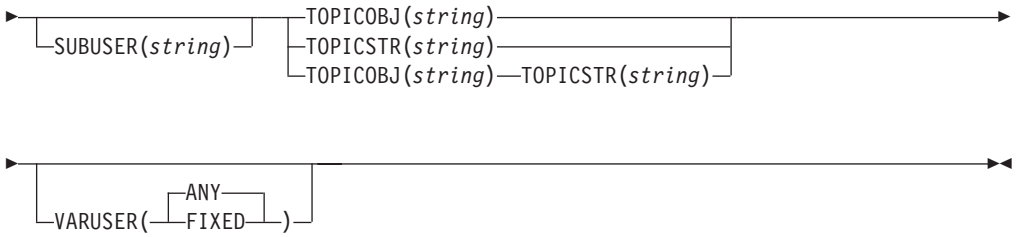
Use DEFINE SUB to allow an existing application to participate in a publish/subscribe application by allowing the administrative creation of a subscription.

Synonym

`DEF SUB`

Syntax





Usage notes

1. You must provide the following information when you define a subscription:
   The SUBNAME
   A destination for messages
   The topic to which the subscription applies
2. You can provide the topic name in the following ways:

   TOPICSTR
   the topic is fully specified as the TOPICSTR attribute.

   TOPICOBJ
   The topic is obtained from the TOPICSTR attribute of the named topic object. The named topic object is retained as the TOPICOBJ attribute of the new subscription. This method is provided to help you enter long topic strings through an object definition.

   TOPICSTR and TOPICOBJ
   The topic is obtained by the concatenation of the TOPICSTR attribute of the named topic object and the value of TOPICSTR (see the MQSUB API specification for concatenation rules). The named topic object is retained as the TOPICOBJ attribute of the new subscription.
3. If you specify TOPICOBJ, the parameter must name a WebSphere MQ topic object. The existence of the named topic object is checked at the time the command processes.
4. You can explicitly specify the destination for messages through the use of the DEST and DESTQMGR keywords.

   You must provide the DEST keyword for the default option of DESTCLAS(PROVIDED); if you specify DESTCLAS(MANAGED), a managed destination is created on the local queue manager, so you cannot specify either the DEST or DESTQMGR attribute.
5. At the time the DEF SUB command processes, no check is performed that the named DEST or DESTQMGR exists.

Parameters

The parameter descriptions apply to DEFINE SUB and ALTER SUB commands, with the following exceptions:

string A mandatory parameter. Specifies the unique name for this subscription, see SUBNAME property.

DEST(*string***)**
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS
System managed destination.

PROVIDED
The destination is a queue.
MANAGED
The destination is managed.

DESTCORL(*string***)**
The CorrelId used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY
The time to expiry of the subscription object from the creation date and time.
integer The time to expiry, in tenths of a second, from the creation date and time.
UNLIMITED
There is no expiry time.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.
NONE
Do not add publish subscribe properties to the message.
MSGPROP
Publish subscribe properties are added as message properties.

PUBACCT(*string***)**
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the AccountingToken field of the MQMD.

PUBAPPID(*string***)**
Identity data passed by the subscriber, for propagation into messages published to this subscription in the ApplIdentityData field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.
ASPUB
Priority of the message sent to this subscription is taken from the priority supplied in the published message.
ASQDEF
Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.
integer An integer providing an explicit priority for messages published to this subscription.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.
NO All publications on the topic are delivered to this subscription.
YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SUBSCOPE
The scope of the subscription.

QMGR
The subscription forwards messages published on the topic only within this queue manager.

QMGR is the only value supported in z/VSE.

SUBUSER(*string***)**
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription.

TOPICSTR(*string***)**
Specifies a fully qualified topic name, or a topic set using wildcard character for the subscription.

TOPICOBJ(*string***)**
The name of a topic object used by this subscription.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.
ANY Any user can connect to and takeover ownership of the subscription.
FIXED
Takeover by another USERID is not permitted.

Delete Subscription

The Delete Subscription (MQCMD_DELETE_SUBSCRIPTION) command deletes a subscription.

Required parameters:

SubName or SubId

Optional parameters:

None

Required parameters

One of:

SubName (MQCFST)
Subscription name (parameter identifier: MQCACF_SUB_NAME). Specifies the unique subscription name. The subscription name, if provided, must be fully specified; a wildcard is not acceptable. The subscription name must refer to a durable subscription.

If SubName is not provided, SubId must be specified to identify the subscription to be deleted. The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier.

You must supply a value for SubId if you have not supplied a value for SubName.

DISPLAY SUB

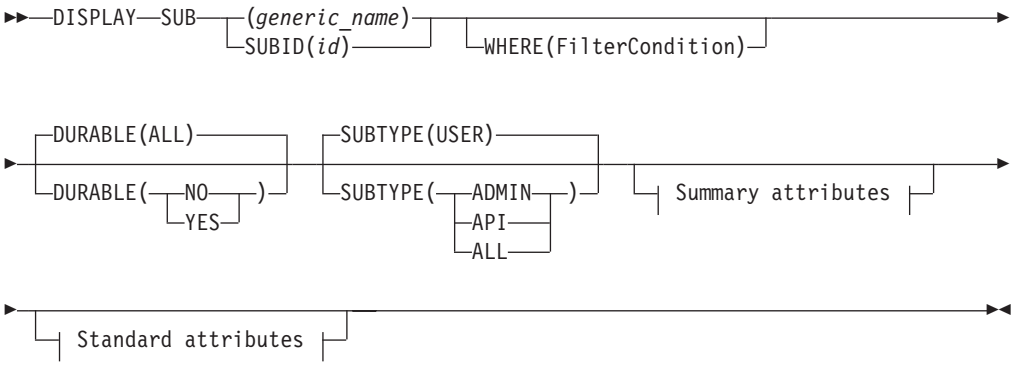
Purpose

Use the MQSC command DISPLAY SUB to display the attributes associated with a subscription.

Synonym

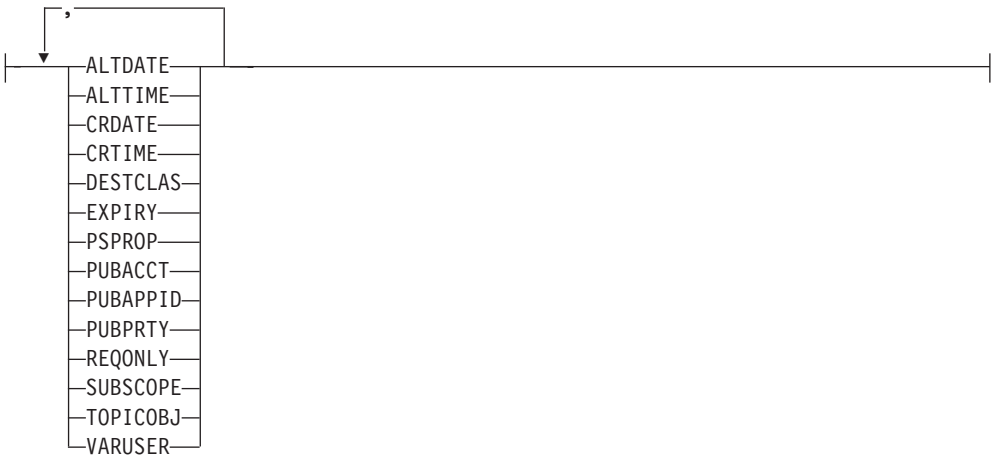
DIS SUB

Syntax



Summary attributes:

Standard attributes:



Usage notes

The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

Parameters

You must specify either the name or the identifier of subscription you want to display. This can be a specific subscription name, or SUBID, or a generic subscription name. By using a generic subscription name, you can display either:

All subscription definitions

One or more subscriptions that match the specified name

The following are valid forms:

```
DIS SUB(XYZ)
DIS SUB SUBID(123)
DIS SUB('xyz*')
```

generic_name
: The local name of the subscription definition to be displayed. A trailing asterisk (*) matches all subscriptions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all subscriptions.

WHERE
: See “WHERE” on page 511 for information about this parameter.

SUMMARY
: Specify this to display the set of summary attributes; this is the default value. This is the default if you do not specify a generic name and do not request any specific attributes.

ALL Specify this to display all the attributes.

If this parameter is specified, any attributes that are also requested specifically have no effect; all attributes are still displayed.

ALTDATE(*string***)**
: The date of the most recent MQSUB or ALTER SUB command that modified the properties of the subscription.

ALTTIME(*string***)**
The time of the most recent MQSUB or ALTER SUB command that modified the properties of the subscription.

CRDATE(*string***)**
The date of the first MQSUB or DEF SUB command that created this subscription.

CRTIME(*string***)**
The time of the first MQSUB or DEF SUB command that created this subscription.

DEST(*string***)**
The destination queue for messages published to this subscription.

DESTCLAS
System managed destination.
PROVIDED
The destination is a queue.
MANAGED
The destination is managed.

DESTCORL(*string***)**
The CorrelId used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.
ALL Display all subscriptions.
NO The subscription is removed when the application that created it, is closed or disconnected from the queue manager.
YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

EXPIRY
The time to expiry of the subscription object from the creation date and time.
integer The time to expiry, in tenths of a second, from the creation date and time.
UNLIMITED
There is no expiry time. This is the default option supplied with the product.

PSPROP
The manner in which publish subscribe related message properties are added to messages published to this subscription.
NONE
Do not add publish subscribe properties to the message.
MSGPROP
Publish subscribe properties are added as message properties.

PUBACCT(*string***)**
Accounting token passed by subscriber, for propagation into messages published to this subscription in the AccountingToken field of the MQMD.

PUBAPPID(*string*)
Identity data passed by the subscriber for propagation into messages published to this subscription in the ApplIdentityData field of the MQMD.

PUBPRTY
The priority of the message published to this subscription. 0 is always returned.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.
NO All publications on the topic are delivered to this subscription.
YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

SUB(*string*)
The application's unique identifier for a subscription.

SUBID(*string*)
The internal, unique key identifying a subscription.

SUBTYPE
Indicates how the subscription was created.
USER Displays only API and ADMIN subscriptions.
ADMIN
Created using DEF SUB MQSC or PCF command. This SUBTYPE also indicates that a subscription has been modified using an administrative command.
API Created using an MQSUB API request.
ALL All.

SUBUSER(*string*)
The user ID that owns this subscription, which can be either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription.

TOPICOBJ(*string*)
The name of a topic object used by this subscription.

TOPICSTR(*string*)
Specifies a fully qualified topic name, or topic set using wildcard character for the subscription.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.
ANY Any user can connect to and takeover ownership of the subscription.
FIXED
Takeover by another USERID is not permitted.

DISPLAY SBSTATUS

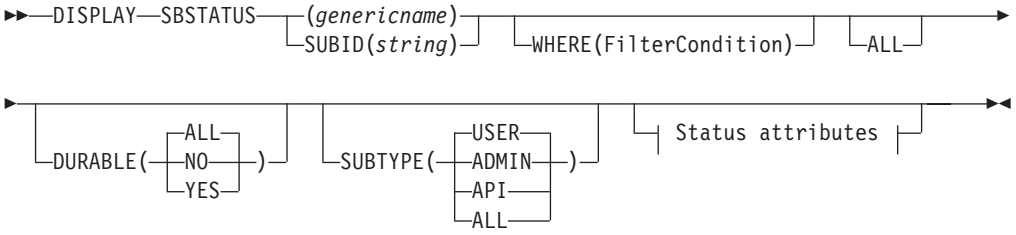
Purpose

Use the MQSC command DISPLAY SBSTATUS to display the status of a subscription.

Synonym

```
DIS SBSTATUS
```

Syntax



Status attributes:

Parameters

You must specify the name of the subscription definition for which you want to display status information. This can be a specific subscription name or a generic subscription name. By using a generic subscription name, you can display either:

All subscription definitions
One or more subscriptions that match the specified name

WHERE
See "WHERE" on page 511 for information about this parameter.

ALL Display all the status information for each specified subscription definition. This is the default if you do not specify a generic name, and do not request any specific parameters.

DURABLE
Specify this attribute to restrict the type of subscriptions which are displayed.
ALL Display all subscriptions.
NO Only information about nondurable subscriptions is displayed.
YES Only information about durable subscriptions is displayed.

SUBTYPE
Specify this attribute to restrict the type of subscriptions which are displayed.
USER Displays only API and ADMIN subscriptions.
ADMIN
Only subscriptions that have been created by an administration interface or modified by an administration interface are selected.
API Only subscriptions created by applications using a WebSphere MQ API call are selected.
ALL All subscription types are displayed (no restriction).

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

ACTCONN
Returns the ConnId of the HConn that currently has this subscription open.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.

NO The subscription is removed when the application that created it is closed or disconnected from the queue manager.

YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

LMSGDATE
The date on which a message was last published to the destination specified by this subscription.

LMSGTIME
The time on which a message was last published to the destination specified by this subscription.

NUMMSGS
The number of messages put to the destination specified by this subscription.

RESMDATE
The date of the most recent MQSUB API call that connected to the subscription.

RESMTIME
The time of the most recent MQSUB API call that connected to the subscription.

SUBID(*string***)**
The internal, unique key identifying a subscription.

SUBTYPE
Indicates how the subscription was created.

ADMIN
Created using the DEF SUB MQSC or PCF command. This SUBTYPE also indicates that a subscription has been modified using an administrative command.

API Created using an MQSUB API call.

For more details of these parameters, see DEFINE SUB

WebSphere MQ Topic

Topic objects can be created, modified, deleted and displayed using MQSC commands. The following MQSC commands are supported:

ALTER TOPIC
DEFINE TOPIC
DELETE TOPIC
DISPLAY TOPIC
DISPLAY TPSTATUS
CLEAR TOPICSTR

ALTER TOPIC

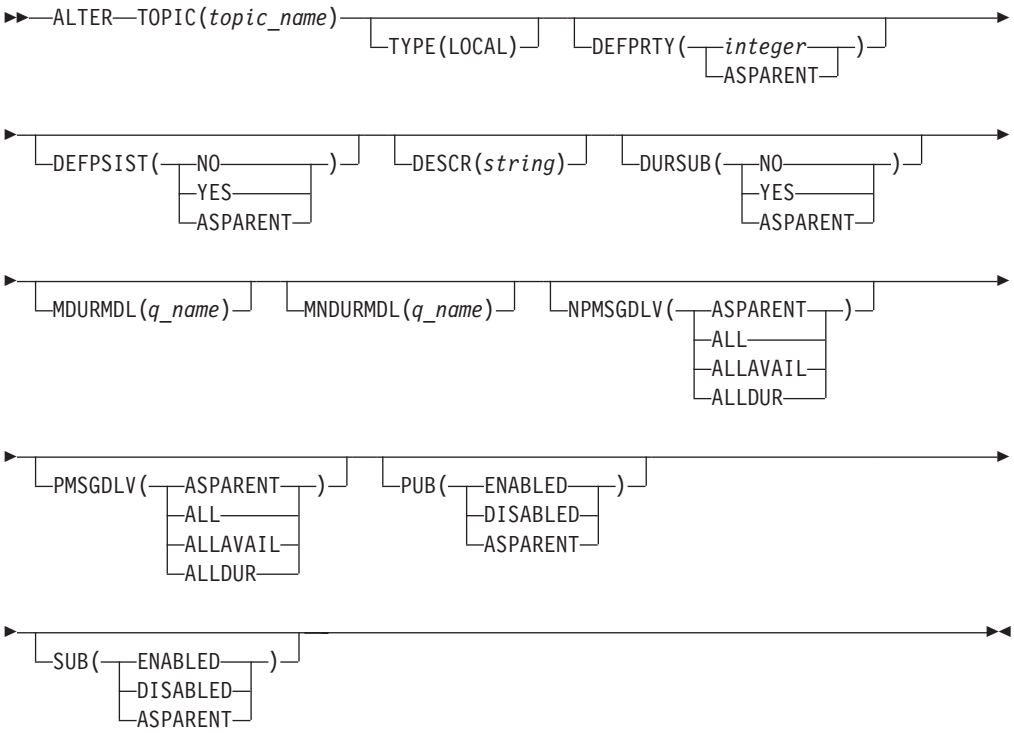
Purpose

Use ALTER TOPIC to alter the parameters of an existing WebSphere MQ topic object.

Synonym

ALT TOPIC

Syntax



Parameters

topic_name

Name of the WebSphere MQ topic definition (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters. This is required.

The name must not be the same as any other topic definition currently defined on this queue manager.

DEFPRTY(*integer***)**

The default priority of messages published to the topic.

integer The value must be in 0 in z/VSE.

DEFPSIST

Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

YES Messages on this queue survive a restart of the queue manager.

Only persistent messages are supported in z/VSE.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY TOPIC command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

MDURMDL(*string***)**
The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters.

If MDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string***)**
The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters.

If MNDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MNDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

SUB Controls whether applications are to be permitted to subscribe to this topic.
ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.
ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).
DISABLED
Applications cannot subscribe to the topic.

TYPE(*topic-type***)**
This is optional. If used it must follow immediately after the topic-name parameter.

LOCAL
A local topic object (only valid value in z/VSE).

DEFINE TOPIC

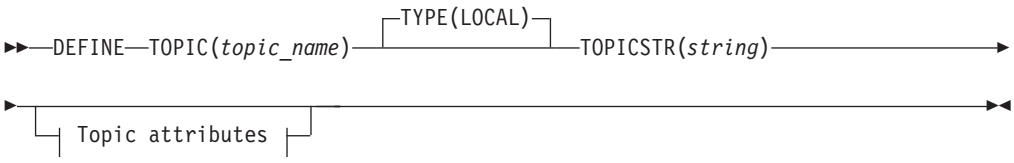
Purpose

Use DEFINE TOPIC to define a new WebSphere MQ administrative topic node in a topic tree, and set its parameters.

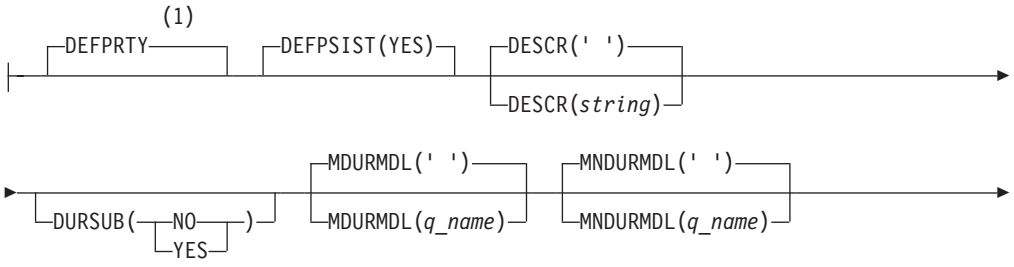
Synonym

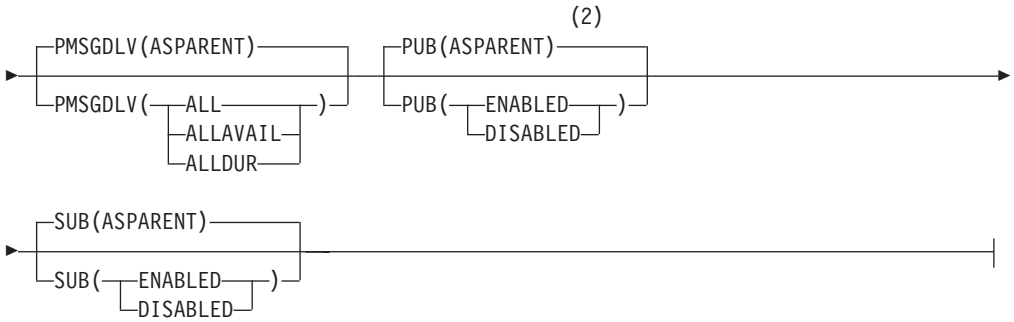
`DEF TOPIC`

Syntax



Topic attributes:





Notes:

1 See Usage note 1.

2 See Usage note 2.

Usage notes

1. When an attribute has the value ASPARENT, the value is taken from the setting of the first parent administrative node that is found in the topic tree. Administered nodes are based on either locally defined topic objects or remotely defined cluster topics when participating in a publish/subscribe cluster. If the first parent topic object also has the value ASPARENT, the next object is looked for. If every object that is found, when looking up the tree, uses ASPARENT, the values are taken from the SYSTEM.BASE.TOPIC, if it exists. If SYSTEM.BASE.TOPIC does not exist, the values are the same as the values supplied with WebSphere MQ in the definitionof the SYSTEM.BASE.TOPIC.
2. When a publication is sent to multiple subscribers, the attributes used from the topic object are used consistently for all subscribers that receive the publication. For example, inhibiting publication on a topic is applied for the next application MQPUT to the topic. A publication that is in progress to multiple subscribers completes to all subscribers. This publication does not take note of a change that has happened, part of the way through, to any attribute on the topic.

Parameters

topic-name

Name of the WebSphere MQ topic definition (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters. This is required.

The name must not be the same as any other topic definition currently defined on this queue manager.

DEFPRTY(*integer***)**

The default priority of messages published to the topic.

integer The value must be in 0 in z/VSE.

DEFPSIST

Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

YES Messages on this queue survive a restart of the queue manager.

Only persistent messages are supported in z/VSE.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY TOPIC command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

MDURMDL(*string***)**
The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters.

If MDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string***)**
The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming WebSphere MQ objects). The maximum length is 48 characters.

If MNDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MNDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

SUB Controls whether applications are to be permitted to subscribe to this topic.
ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree. This is the default supplied with WebSphere MQ, but your installation might have changed it.
ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).
DISABLED
Applications cannot subscribe to the topic.

TOPICSTR(*string***)**
The topic string represented by this topic object definition. This parameter is required and cannot contain the empty string.

The topic string must not be the same as any other topic string already represented by a topic object definition.

The maximum length of the string is 256 characters in z/VSE.

TYPE(*topic-type***)**
This is optional. If used it must follow immediately after the topic-name parameter.

LOCAL
A local topic object (only valid value in z/VSE).

DELETE TOPIC

Purpose

Use DELETE TOPIC to delete a WebSphere MQ administrative topic node.

Synonym

None

Syntax

Parameters

topic_name
The name of the administrative topic object to be deleted. This parameter is required. The name must be that of an existing administrative topic object.

DISPLAY TOPIC

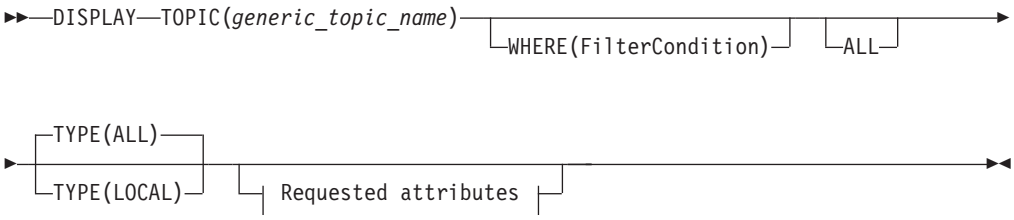
Purpose

Use the MQSC command DISPLAY TOPIC to display the attributes of one or more WebSphere MQ topic objects of any type.

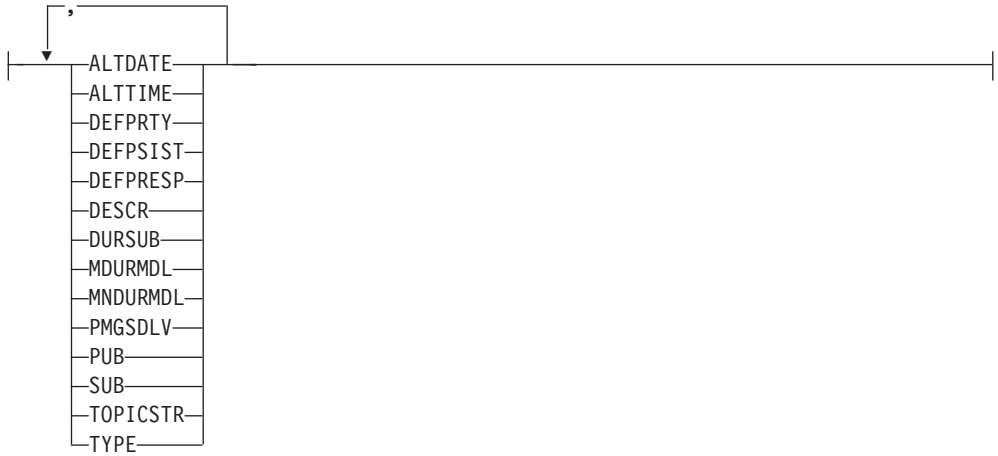
Synonym

DIS TOPIC

Syntax



Requested attributes:



Usage notes

The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

Parameters

You must specify the name of the topic definition you want to display. This can be a specific topic name or a generic topic name. By using a generic topic name, you can display either:

- All topic definitions
- One or more topic definitions that match the specified name

generic-topic-name
The name of the administrative topic definition to be displayed (see Rules for naming WebSphere MQ objects). A trailing asterisk (*) matches all administrative topic objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all administrative topic objects.

WHERE
See “WHERE” on page 511 for information about this parameter.

ALL Specify this to display all the attributes. If this parameter is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default if you do not specify a generic name, and do not request any specific attributes.

TYPE Specifies the type of topics that you want to be displayed. Values are:
ALL Display all topic types (same as LOCAL in z/VSE).
LOCAL
Display locally defined topics.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

ALTDATE
The date on which the definition or information was last altered, in the form yyyy-mm-dd.

ALTTIME
The time at which the definition or information was last altered, in the form hh.mm.ss.

DEFPRTY
Default priority of the messages published to this topic.

DEFPSIST
Default persistence of messages published to this topic.

DEFPRESP
Default put response for this topic. This attribute defines the behavior that should be used by applications when the put response type in the MQPMO options has been set to MQPMO_RESPONSE_AS_TOPIC_DEF.

DESCR
Description of this administrative topic object.

DURSUB
Whether the topic permits durable subscriptions to be made.

MDURMDL
The name of the model queue for durable managed subscriptions.

MNDURMDL
The name of the model queue for non-durable managed subscriptions.

PMSGDLV
The delivery mechanism for persistent messages.

PUB Whether the topic is enabled for publication.

SUB Whether the topic is enabled for subscription.

TOPICSTR
The topic string.

DISPLAY TPSTATUS

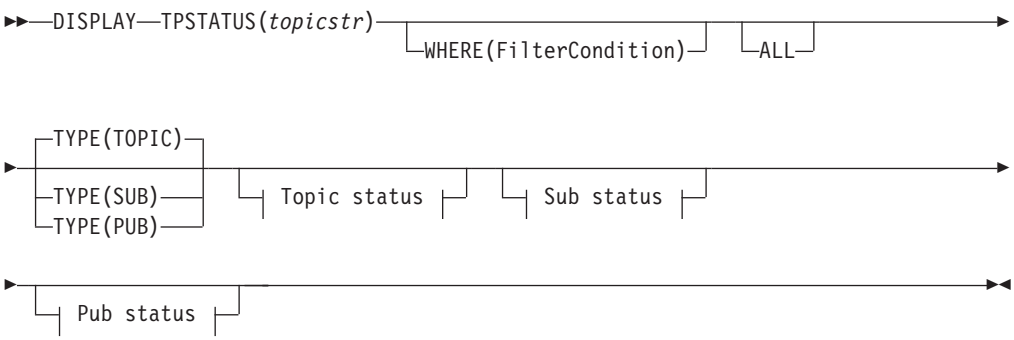
Purpose

Use the MQSC command DISPLAY TPSTATUS to display the status of one or more topic nodes in a topic tree.

Synonym

```
DIS TPS
```

Syntax



Topic status:

Sub status:

Pub status:

Parameters

The DISPLAY TPSTATUS (DIS TPS) command displays the status of one or more topic nodes in a topic tree, as specified by the parameters supplied.

The DISPLAY TPSTATUS command requires a topic string value to determine which topic nodes the command returns.

generic-topic-name
The name of the administrative topic definition to be displayed (see Rules for naming WebSphere MQ objects). A trailing asterisk (*) matches all administrative topic objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all administrative topic objects.

WHERE
See "WHERE" on page 511 for information about this parameter.

ALL Use this parameter to display all attributes.

If this parameter is specified, any attributes that you request specifically have no effect; the command displays all attributes.

This is the default parameter if you do not specify a generic name, and do not request any specific attributes.

TYPE

TOPIC
The command displays status information relating to each topic node, which is the default if you do not provide a TYPE parameter.

PUB The command displays status information relating to applications that have topic nodes open for publish.

SUB The command displays status information relating to applications that subscribe to the topic node or nodes. Note that the subscribers that the command returns are not necessarily the subscribers that would receive a message published to this topic node.

Topic status parameters

Topic status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ADMIN
If the topic node is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

DEFPRTY
Displays the resolved default priority of messages published to the topic (always 0 in z/VSE)

DURSUB
Displays the resolved value that shows whether applications can make durable subscriptions. The value can be YES or NO.

MDURMDL
Displays the resolved value of the name of the model queue to be used for durable subscriptions.

MNDURMDL
Displays the resolved value of the name of the model queue used for non-durable subscriptions

PMSGDLV
Displays the resolved value for the delivery mechanism for persistent messages published to this topic. The value can be ALL, ALLDUR, or ALLAVAIL, but not ASPARENT.

PUB Displays the resolved value that shows whether publications are allowed for this topic, if there is no ASPARENT response value. The value can be ENABLED or DISABLED.

PUBCOUNT
Displays the number of handles that are open for publish on this topic node.

RETAINED
Displays whether there is a retained publication associated with this topic. The value can be YES or NO.

SUB Displays the resolved value that shows whether subscriptions are allowed for this topic, if there is no ASPARENT response value. The values can be ENABLED or DISABLED.

SUBCOUNT
Displays the number of subscribers to this topic node, including durable subscribers that are not currently connected.

Sub status parameters

Sub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
Detects local publications, returning the currently active ConnectionId (CONNID) that opened this subscription.

DURABLE
Indicates whether a durable subscription is not deleted when the creating application closes its subscription handle, and persists over queue manager restart. The value can be YES or NO.

LMSGDATE
The date on which an MQPUT call last sent a message to this subscription. The MQPUT call updates the date field only when the call successfully puts a message to the destination specified by this subscription. Note that an MQSUBRQ call causes an update to this value.

LMSGTIME
The time at which an MQPUT call last sent a message to this subscription. The MQPUT call updates the time field only when the call successfully puts a message to the destination specified by this subscription. Note that an MQSUBRQ call causes an update to this value.

NUMMSGS
Number of messages put to the destination specified by this subscription. Note that an MQSUBRQ call causes an update to this value.

RESMDATE
Date of the most recent MQSUB call that connected to this subscription.

RESMTIME
Time of the most recent MQSUB call that connected to this subscription.

SUBID
An all time unique identifier for this subscription, assigned by the queue manager. The format of SUBID matches that of a CorrelId. For durable subscriptions, the command returns the SUBID even if the subscriber is not currently connected to the queue manager.

SUBTYPE
The type of subscription, indicating how it was created. The value can be ADMIN, or API.

SUBUSER
The user ID that owns this subscription, which can be either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription.

Pub status parameters

Pub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
The currently active ConnectionId (CONNID) associated with the handle that has this topic node open for publish.

LPUBDATE
The date on which this publisher last sent a message.

LPUBTIME
The time at which this publisher last sent a message.

NUMPUBS
Number of publishes by this publisher. Note that this value records the actual number of publishes, not the total number of messages published to all subscribers.

CLEAR TOPICSTR

Purpose

Use the MQSC command CLEAR TOPICSTR to clear the retained message which is stored for the specified topic string.

Synonym

None

Syntax

Usage notes

If the topic string specified has no retained message the command completes successfully. You can find out whether a topic string has a retained message by using the DISPLAY TPSTATUS command. The RETAINED field shows whether there is a retained message.

The topic-string input parameter on this command must match the topic you want to act on. You are advised to keep the character strings in your topic strings as characters that can be used from location issuing the command. If you issue commands using MQSC, there are fewer characters available to you than if you are using an application submitting PCF messages, such as the WebSphere MQ Explorer.

Parameters

You must specify which topic string you want to remove the retained publication from.

topic_string
The topic string to be cleared. This string can represent several topics to be cleared by using wildcard character #.

In z/VSE only a trailing wildcard is allowed. This selects any child topics.

For example, specifying `TOPICSTR('/a/bb/#')` clears the following topics:

```
/a/bb
/a/bb/z
/a/bb/c/z
/a/bb/c/y/z
```

CLRTYPE
This is a mandatory parameter. The value must be:
RETAINED
Remove the retained publication from the specified topic string.

SCOPE
The scope of the deletion of retained messages. The value can be:
LOCAL
The retained message is removed from the specified topic string at the local queue manager only. This is the default value.

WebSphere MQ meta commands

The MQPMQSC utility program also supports the following meta commands:
- SDEFS
- COMMAND

The SDEFS command

The SDEFS command can be used to produce a set of DEFINE commands for locally defined objects. If a DLBL with filename MQSDEFS for a sequential file is specified then the output will be written to it, otherwise the output is directed to SYSLST.

For example:

```
// DLBL MQSDEFS,'sam.file',0,VSAM,CAT=VSEMQUC,RECSIZE=80,     X
               RECORDS=(250,100),DISP=(NEW,KEEP,DELETE)
// EXEC MQPMQSC,SIZE=MQPMQSC
SDEFS OBJECT(QLOCAL)
/*
```

In this example, the SDEFS command, with OBJECT(QLOCAL), generates one DEFINE statement for each local queue defined to the local queue manager, and writes the statements to the MQSDEFS file.

Multiple SDEFS in the SYSIPT stream for the MQPMQSC utility results in generated statements being appended sequentially to the MQSDEFS file.

The SDEFS command can be used to generate define commands for the following OBJECTs:

OBJECT(QMGR)
OBJECT(QUEUE)
OBJECT(QLOCAL)
OBJECT(QMODEL)
OBJECT(QALIAS)
OBJECT(QREMOTE)
OBJECT(CHANNEL)
OBJECT (NAMELIST)

In the case of OBJECT(QMGR), an ALTER command is generated. The OBJECT(QUEUE) option generates DEFINE commands for all queue types.

The commands generated by the SDEFS command can be used as input to the MQPMQSC utility to define the same objects.

Note: DEFINE commands are not generated for any local queues that can be identified as dynamic.

The COMMAND command

The COMMAND command tells the MQPMQSC utility to read its input (MQSC commands) from a SAM file whose filename is specified in the DDNAME parameter.

For example:

```
// DLBL filenam,'sam.file',0,VSAM,CAT=VSEMQUC,RECSIZE=80,     X
               RECORDS=(250,100),DISP=(OLD,KEEP,DELETE)
// EXEC MQPMQSC,SIZE=MQPMQSC
COMMAND DDNAME(filenam)
/*
```

You can specify multiple COMMAND statements in the SYSIPT stream. In addition, you can mix MQSC commands and COMMAND statements in the SYSIPT stream.

For example:

```
// EXEC MQPMQSC,SIZE=MQPMQSC
ALTER QMGR CCSID(1047)
COMMAND DDNAME(filenm1)
COMMAND DDNAME(filenm2)
DISPLAY QMGR CCSID
/*
```

The SYSIPT stream is processed sequentially.

Note: DEFINE and ALTER commands for queues may fail if the attribute values for the queue fall outside the limitations of the queue manager.

You should not specify SDEFS and COMMAND for the same file in the same job step since the SDEFS file is not closed until end of the job step.

Chapter 10. WebSphere MQ clients

This chapter describes the WebSphere MQ for z/VSE client/server environment, including an overview of the WebSphere MQ client, installation and administration instructions, and application programming information relevant to client applications.

Introduction to WebSphere MQ clients

This section provides the following information:
- WebSphere MQ client overview.
- Purpose of WebSphere MQ clients.

WebSphere MQ client overview

An WebSphere MQ client is part of the WebSphere MQ product that can be installed on its own, on a separate machine from the base product and server.

You can run an WebSphere MQ application on an WebSphere MQ client and it can interact with one or more WebSphere MQ servers and connect to their queue managers by means of a communications protocol. The servers to which the client connects might or might not be part of a cluster.

The WebSphere MQ MQI is available to applications running on the client platform; the queues and other WebSphere MQ objects are held on a queue manager that you have installed on a server machine.

An application that you want to run in the WebSphere MQ client environment must first be linked with the relevant client library. When the application issues an MQI call, the WebSphere MQ client directs the request to a queue manager, where it is processed and from where a reply is sent back to the WebSphere MQ client.

The link between the application and the WebSphere MQ client is established dynamically at runtime.

An application running in the WebSphere MQ client environment runs in synchronous mode because there must be an active connection between the client and server machines.

The connection is made by an application issuing an MQCONN or MQCONNX call. Clients and servers communicate through MQI channels. When the call succeeds, the MQI channel remains connected until the application issues a MQDISC call. This is the case for every queue manager that an application needs to connect to.

You can also run an application in the WebSphere MQ client environment when your machine also has a queue manager installed. In this situation, you have the choice of linking to the queue manager libraries or the client libraries, but remember that if you link to the client libraries, you still need to define the channel connections. This can be useful during the development phase of an application. You can test your program on your own machine, with no dependency on others, and be confident that it will still work when you move it to a full WebSphere MQ environment.

The client MQI must be called by an LE-conforming program, i.e. one compiled with C for z/VSE, COBOL for z/VSE, or PLI for z/VSE. This requirement means client applications must run in a z/VSE environment that supports Language Environment for z/VSE.

For applications that do not run in an environment that supports Language Environment for z/VSE, this WebSphere MQ client for z/VSE includes a client Bridge facility. The client Bridge includes a long-running CICS transaction that accepts connections from applications using the Bridge MQI (provided with the client). Connections to the Bridge are established using the MQCONNX MQI call. The Bridge transaction co-ordinates "proxy" transactions in CICS that issue client MQI calls on behalf of the non-LE client application.

Purpose of WebSphere MQ clients

Using WebSphere MQ clients is an efficient way of implementing WebSphere MQ messaging and queuing. You can have an application that uses the MQI running on one machine and the queue manager running on a different machine (either physical or virtual).

The benefits of doing this are:

- There is no need for a full WebSphere MQ implementation on the client machine.
- Hardware requirements on the client system are reduced.
- System administration requirements are reduced.
- An WebSphere MQ application running on a client can connect to multiple queue managers on different systems.

Installing WebSphere MQ clients

This section provides the following information:

- Prerequisites for the WebSphere MQ client.
- Installing WebSphere MQ client components.
- Configuring communication links.
- Verifying the installation.
- WebSphere MQ for z/VSE client differences.

Prerequisites for the WebSphere MQ client

The WebSphere MQ client environment is incomplete without at least one WebSphere MQ server product running on an accessible system. The WebSphere MQ server can be running on the same system as the client or on a remote system. The server can run on any MQ platform that supports MQ client connections.

In addition, the WebSphere MQ client for z/VSE has the following software prerequisite:

- z/VSE 3.1, or later.
- Language Environment for z/VSE 1.4.4 or later with:
  - PK05874/UK03996 for 1.4.4
- TCP/IP for VSE 1.5F (or equivalent) or later.

The WebSphere MQ client for z/VSE functions in the z/VSE batch environment as well as:

- CICS for z/VSE 2.3
- CICS Transaction Server 1.1.1 (or later)

Installing WebSphere MQ client components

The WebSphere MQ client for z/VSE is available as an IBM SupportPAC (MQC5), which can be downloaded from the IBM web site.

The MQC5 SupportPac is provided in a single ZIP file, MQC5.ZIP. Once the SupportPac has been downloaded, to install the WebSphere MQ client for z/VSE, carry out the following instructions:

1. Unzip the MQC5.ZIP file to a (new) directory of your choice (which can be temporary); for example, 'mqc5'.

   Unzipping the MQC5.ZIP file should create or replace the following files:
   MQICLIB.JCL
   MQICINC.JCL
   MQICSMP.JCL
   MQICBRG.JCL
   README.TXT
   LICENSES*.TXT
2. Copy the four .JCL files (MQICLIB, MQICINC, MQICSMP and MQICBRG) to your z/VSE system. These are binary files, so if you are using FTP, for example, you should specify 'binary' rather than 'ascii' to avoid ASCII to EBCDIC translation.
3. Modify the JCL files as necessary for your system, for example, add POWER job cards as relevant. Each JCL file identifies a target VSE sublibrary for installation via a // SETPARM card. You can modify the SETPARM card to identify your target installation sublibrary, or specify the SETPARM value at job run time.

   Note: If you already have the WebSphere MQ for z/VSE product installed, you should not install the MQ Client for z/VSE SupportPac into the same sublibrary. Choose a different sublibrary. The default installation sublibrary is PRD2.MQICVSE.
4. Submit the MQICLIB job. This job creates your installation sublibrary if it does not already exist, so it should be run before you run the other JCL files. It also catalogs and link-edit WebSphere MQ client for z/VSE object and phase files.

   Jobs that run your MQ client applications must include the installation sublibrary in their LIBDEF SEARCH path. If you are running MQ client applications in CICS, the LIBDEF SEARCH path of your CICS startup JCL should include the installation sublibrary.
5. Submit the MQICINC job. This job catalog3 the copybooks and header files needed to write your own MQ client applications. Copybooks and header files are available for C for z/VSE, COBOL for z/VSE and PLI for z/VSE.

   Jobs that compile your MQ client applications must include the installation sublibrary in their LIBDEF SEARCH path.
6. Submit the MQICSMP job (optional). This job catalogs sample MQ client source code programs. Samples are provided for C for z/VSE, COBOL for z/VSE and PLI for z/VSE, and include the following:

   COBOL/VSE Samples

   MQICGETC
   Get a message from a queue.

   MQICPUTC
   Put a message on a queue.

   MQICINQC
   Inquire on a queue's attributes.

MQICSETC
Set a queue's attributes.

C/VSE Samples
MQICCNXX
Connect to two queue managers simultaneously.
MQICGETX
Get a message from a queue.
MQICPUTX
Put a message on a queue.
MQICINQX
Inquire on a queue's attributes.
MQICSETZ
Set a queue's attributes.

PLI/VSE Samples
MQICGETP
Get a message from a queue.
MQICPUTP
Put a message on a queue.

7. Submit the MQICBRG job (optional). This job creates components of the WebSphere MQ client for z/VSE bridge in your installation sublibrary. The client bridge provides a means to run non-LE client applications in the z/VSE environment. Such applications issue MQI calls like normal client programs, but the MQI calls are performed by 'partner' transactions in CICS, managed by the bridge.

To uninstall the WebSphere MQ client for z/VSE, delete the installation sublibrary.

Configuring communication links

The WebSphere MQ client uses a communication link between the client application and the MQ server. In WebSphere MQ, these communication links are called channels. The channels used to connect WebSphere MQ clients to servers are called MQI channels.

You set up channel definitions at each end of your link so that your WebSphere MQ application on the WebSphere MQ client can communicate with the queue manager on the server.

The WebSphere MQ client for z/VSE supports TCP/IP channels only.

WebSphere MQ server configuration

The server can run in any environment that supports WebSphere MQ and client connections, for example, z/VSE, AIX®, Linux, z/OS.

Configuration for the server environment can be specific, so it is necessary to refer to relevant WebSphere MQ documentation for the specific platform that hosts the WebSphere MQ server.

Generally, the server environment requires:

An active queue manager
The server queue manager must be active and available for MQI requests. This is because the server performs the client's MQI requests as a proxy, and return the results of the client MQI requests to the client application.

An active TCP/IP subsystem
Since the WebSphere MQ client for z/VSE only supports TCP/IP channels, a TCP/IP subsystem must be active and available on the server system. If not, the client is not able to establish a connection with the MQ server.

An active MQ listener task
The MQ listener task waits for and accepts client connection requests. The connection, and subsequent client conversation, is handled by the MQ server. The MQ listener is bound to an IP port number (port 1414 by default). The port number is specified by the client when it attempts to establish a connection to the MQ server. If an MQ listener task is not active, the client is not able to establish a connection with the MQ server.

A server-connection channel definition
A server-connection (SVRCONN) channel definition provides information and parameters that describe and control the client/server connection. A valid SVRCONN channel must be defined on the server queue manager. The client application names this channel and, if required, provide a matching client-connection (CLNTCONN) channel definition when it attempts to establish a connection with the MQ server.

To define a server-connection channel on WebSphere MQ for z/VSE, refer to "Channel definitions" on page 112.

WebSphere MQ client configuration

The WebSphere MQ client environment requires:

An active TCP/IP subsystem
Like the server, the WebSphere MQ client for z/VSE requires an active TCP/IP subsystem. The TCP/IP subsystem must be active and available in the client environment. If not, the client is not able to establish a connection with the MQ server.

A client-connection channel definition
A client-connection (CLNTCONN) channel definition provides information and parameters that describe and control the client/server connection. The name of the CLNTCONN channel must match the name of the SVRCONN channel defined on the server queue manager. There are two ways an WebSphere MQ client can provide the client-connection channel definition:

- Using the MQSERVER environment variable in conjunction with the MQCONN MQI call.
- Using the MQCNO and MQCD data structures in conjunction with the MQCONNX MQI call.

Each of these approaches is described in this chapter. Both methods provide details of the server's IP address and the MQ listener's port number. This way, the MQCONN or MQCONNX call can establish a client connection with an intended MQ Server.

Verifying the installation

You can verify your WebSphere MQ client/server environment using the supplied sample programs. These verify that your installation has been completed successfully and that the communication link is working.

The WebSphere MQ client environment cannot be verified until the client has been installed and a server environment has been configured. These tasks are described in the preceding sections.

To verify the WebSphere MQ client/server environment, it is necessary to perform these tasks:

1. **Build the sample programs**

   Any of the supplied samples can be used to verifying the WebSphere MQ client/server environment. A sample that puts a message, and another that gets a message, verify the core MQI calls: MQCONN, MQOPEN, MQPUT, MQGET, MQCLOSE and MQDISC. For this reason, the COBOL samples MQICPUTC and MQICGETC are recommended. Instructions on how to build these samples are provided later in this chapter.

2. **Run the sample programs**

   Once the sample programs have been built, run the samples as batch jobs to verify the WebSphere MQ client/server environment. If the samples MQICPUTC and MQICGETC are used, you can expect the messages put by MQICPUTC to be retrieved and displayed by MQICGETC. Instructions on how to run these samples are provided later in this chapter.

WebSphere MQ for z/VSE client differences

WebSphere MQ client function is documented in the WebSphere MQ Clients manual (GC34-6590). This manual described function for Version 6 MQ clients, however, the function is generally the same for z/VSE MQ clients, with the following additional points:

- The WebSphere MQ client for z/VSE does not support channel tables. Instead, client applications should use the MQSERVER environment variable to identify a channel name, or use the MQCONNX MQI call and specify client channel information via the MQCNO and MQCD data structures.
- The WebSphere MQ client for z/VSE does not support SSL enabled channels. For security, client channels can use security, send and receive exits.
- The WebSphere MQ client for z/VSE supports TCP/IP channels only.
- The WebSphere MQ client for z/VSE uses the MQDATA environment variable to identify a target for client log messages (e.g. client application error messages).

  The MQDATA environment variable, if used in batch, should name a DLBL of a VSAM managed SAM file, or an ESDS. Messages lengths are variable up to 100 characters. If used in CICS, the MQDATA variable should name a CICS transient data queue. If the MQDATA variable is not set, client log messages are discarded.
- The WebSphere MQ client for z/VSE uses the MQTRACE environment variable to identify a target for diagnostic trace messages.

  The MQTRACE environment variable, if used in batch, should name a DLBL of a VSAM managed SAM file, or an ESDS. Messages lengths are variable up to 80 characters. If used in CICS, the MQTRACE variable should name a CICS transient data queue. If the MQTRACE variable is not set, diagnostic trace messages are discarded.

  Client tracing can introduce significant processing overhead for client applications, and should not be used except in conjunction with IBM service for problem resolution.
- The WebSphere MQ client for z/VSE supports the following environment variables:

  MQCCSID
  MQDATA
  MQSERVER
  MQTRACE
  MQ_USER_ID

MQ_PASSWORD

These can be set using the setenv() C run-time call. The setenv() call is only available to C language programs. To avoid this limitation, the WebSphere MQ Client for z/VSE provides two additional calls as follows:

```
void MQENTRY MQSETENV (PMQCHAR  Name,      /* Envir var name */
       PMQCHAR  Value,     /* Envir var value */
       MQLONG   Length,    /* Length of var value */
       PMQLONG  CompCode,  /* Completion code */
       PMQLONG  Reason);   /* Reason code */
```

and

```
void MQENTRY MQGETENV (PMQCHAR  Name,      /* Envir var name */
       PMQCHAR  Value,     /* Envir var value */
       PMQLONG  Length,    /* Length of var value */
       PMQLONG  CompCode,  /* Completion code */
       PMQLONG  Reason);   /* Reason code */
```

These calls use OS linkage, and can be called, for example, as follows:

```
CALL 'MQSETENV' USING WS-VAR-NAME
      WS-VAR-VALUE
      WS-VAL-LEN
      WS-COMPCODE
      WS-REASON.
```

In addition, MQ client environment variables can be set using the Language Environment for z/VSE run-time option ENVAR. In batch, in conjunction with ENVAR, you can use the LE variable _CEE_ENVFILE to set one or more environment variables in a single file. The _CEE_ENVFILE variable is not supported in CICS.

- When building client applications, for CICS or batch, you must prelink your application with the WebSphere MQ Client for z/VSE MQI object file, MQICVSE.OBJ. For example:

```
// OPTION CATAL
   PHASE YOURPROG,*
   INCLUDE YOURPROG
   INCLUDE MQICVSE
// EXEC EDCPRLK
```

Note: MQ client for z/VSE applications must be prelinked.

The MQICVSE.OBJ file resides in the client installation sublibrary. It contains an entry point for each of the support MQI calls. These include:

MQCONN
MQCONNX
MQDISC
MQOPEN
MQCLOSE
MQPUT
MQPUT1
MQGET
MQINQ
MQSET
MQCMIT
MQBACK

Note: WebSphere for z/VSE client does not support message properties.

These MQI calls, at run time, dynamically load and call entry points in the WebSphere MQ client for z/VSE functional phase, MQICVSEP.PHASE. The MQICVSEP phase resides in the client installation sublibrary, and is flagged SVA-eligable.

- When using the MQCONNX client MQI call, the WebSphere MQ client for VSE supports up to version 2 MQCNO data structures, and up to version 4 MQCD data structures.

System administration for clients

This section describes system administration for WebSphere MQ clients, specifically in regard to:

- WebSphere MQ client security.
- Client and server connection channels.
- WebSphere MQ client environment variables.

WebSphere MQ client security

It is important to consider WebSphere MQ client security, so that the client applications do not have unrestricted access to resources on the server.

There are two aspects to security between a client application and its queue manager server:

- Authentication
- Access control

Authentication

Authentication deals with the identity of the client user. The identify of the client user can be determined in two ways:

Environment variables

Environment variables that identify the client user include:

- MQ_USER_ID
- MQ_PASSWORD

These environment variables can be set in the WebSphere MQ client application environment. Their values are passed to the server which can use them to authenticate the client user, or pass them to a channel security exit to authenticate the user.

Channel security exits

The channel security exits for client to server communication can work in the same way as for server to server communication. A pair of exits provide mutual authentication of both the client and the server. A full description is given in the WebSphere MQ Intercommunication manual.

In client to server communication, the channel security exits do not have to operate as a pair. The exit on the WebSphere MQ client side can be omitted. In this case the user ID is placed in the channel descriptor (MQCD) and the security exit can alter it, if required.

Access control

Access control deals with access permissions to WebSphere MQ objects for an authenticated client user.

Access control in WebSphere MQ is based upon the user identifier associated with the process making MQI calls. For WebSphere MQ clients, the process that issues the MQI calls is the server MCA.

The user identifier used by the server MCA is that contained in the MCAUserIdentifier field of the MQCD. The contents of MCAUserIdentifier are determined by:

- Any values set by security exits.
- The MQ_USER_ID environment variable from the client.
- The MCAUSER (in the server-connection channel definition).

Depending upon the combination of settings of the above, MCAUserIdentifier is set to the appropriate value. If a server security exit is provided, MCAUserIdentifier can be set by the exit. Otherwise MCAUserIdentifier is determined as follows:

- If the server-connection channel MCAUSER attribute is nonblank, this value is used.
- If the server-connection channel MCAUSER attribute is blank, the user ID received from the client is used. However, for the clients that use the MQ_USER_ID environment variable to supply the user ID, it is possible that no environment variable is set. In this case, the user ID that started the server channel is used.

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 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 WebSphere MQ objects.

Client and server connection channels

A channel is a logical communication link between an WebSphere MQ client and an WebSphere MQ server, or between two WebSphere MQ servers. A channel has two definitions: one at the sender end of the connection, and one at the receiver end.

The same channel name must be used at each end of the connection, and the channel type used must be compatible.

Types of channels

There are two categories of channel in WebSphere MQ, with different channel types within these categories:

Message channels

A message channel is a one-way link. It connects two queue managers via message channel agents (MCAs). Its purpose is to transfer messages from one queue manager to another.

Message channels are not required by the client server environment.

More information on message channels can be found in the WebSphere MQ Intercommunication manual.

MQI channels

An MQI channel connects an WebSphere MQ client to a queue manager on a server machine, and is established when you issue an MQCONN or MQCONNX call. It is a two-way link and is used for the transfer of MQI calls and responses only, including MQPUT calls that contain message data.

An MQI channel can be used to connect a client to a single queue manager, or to a queue manager that is part of a queue-sharing group.

There are two channel types for MQI channel definitions. They define the bi-directional MQI channel.

Client-connection channel
This type is for the WebSphere MQ client.

Server-connection channel
This type is for the server running the queue manager, with which the WebSphere MQ application, running in an WebSphere MQ client environment, communicates.

Defining MQI channels

To create a new channel, you have to create two channel definitions, one for each end of the connection, using the same channel name and compatible channel types. In this case, the channel types are server-connection and client-connection.

For the server-connection channel, the WebSphere MQ Version 5.1 (and later) products include a feature that can automatically create a channel definition on the server if one does not exist. If an inbound attach request is received from a client and an appropriate server-connection definition cannot be found in the channel definition table, WebSphere MQ creates a definition automatically and adds it to the channel definition table.

Automatic definitions are based on two default definitions supplied with WebSphere MQ: SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN. You enable automatic definition of server-connection definitions by updating the queue manager object using MQSC ALTER QMGR (or the PCF command Change Queue Manager).

For more information about the automatic creation of channel definitions, see the *WebSphere MQ Intercommunication* manual.

Where the server does not automatically define channels, the server-connection channel must be defined manually using MQSC DEFINE CHANNEL (or the PCF command Create Channel).

The client-connection channel cannot be automatically defined or predefined in the WebSphere MQ client for z/VSE environment. The client channel must be named by the MQSERVER environment variable, or described in the MQCD data structure in conjunction with the MQCONNX MQI call.

Client channel definition table

The WebSphere MQ client for z/VSE does not support the channel definition table available in some other MQ client environments. Consequently it is not possible to predefine client-connection channels on z/VSE.

Instead, as already mentioned, the client channel must be named by the MQSERVER environment variable, or described in the MQCD data structure in conjunction with the MQCONNX MQI call.

Channel exits

The channel exits available to the WebSphere MQ client for z/VSE are:
- Send exit
- Receive exit
- Security exit

These exits are available at both the client and the server end of the channel. Exits are not available to your application if you are using the MQSERVER environment variable. Exits are explained in the WebSphere MQ Intercommunication manual.

The send and receive exit work together. There are several possible ways in which you can use them:
- Splitting and reassembling a message.
- Compressing and decompressing data in a message.
- Encrypting and decrypting user data.
- Journaling each message sent and received.

You can use the security exit to ensure that the WebSphere MQ client and server machines are correctly identified, as well as to control access to each machine.

Exit programs in the WebSphere MQ client for z/VSE environment are found via the LIBDEF SEARCH specification.

WebSphere MQ client environment variables

The function of MQI calls when issued by WebSphere MQ client applications can be affected by environment variables and their values.

The WebSphere MQ client environment variables are:

MQCCSID
This variable specifies the coded character set number to be used and overrides the z/VSE default, 500.

MQDATA
This variable identifies a target for client log messages (e.g. client application error messages).

The MQDATA environment variable, if used in batch, should name a DLBL of a VSAM-managed SAM file, or an ESDS. Messages lengths are variable up to 100 characters. If used in CICS, the MQDATA variable should name a CICS transient data queue. If the MQDATA variable is not set, client log messages are discarded.

MQSERVER
This variable is used to define a minimal channel. It specifies the location of the WebSphere MQ server and the communication method to be used. The format of the value for this variable is:

```
MQSERVER=ChannelName/TCP/ConnectionName
```

Note that *ConnectionName* must be a fully-qualified network name; for example, hostname(port), or 1.11.1.11(1414). *ChannelName* cannot contain the forward slash (/) character because it is used to separate the channel name, transport type, and connection name. For z/VSE clients, the transport type must be "TCP".

When the MQSERVER environment variable is used to define a client channel, a maximum message length (MAXMSGL) of 4 MB is used, so larger messages cannot flow across this channel. For larger messages, a client-connection channel must be defined using the MQCD data structure in conjunction with the MQCONNX MQI call, with MAXMSGL set to a larger figure.

MQTRACE
This variable identifies a target for diagnostic trace messages.

The MQTRACE environment variable, if used in batch, should name a DLBL of a VSAM-managed SAM file, or an ESDS. Messages lengths are variable up to 80 characters. If used in CICS, the MQTRACE variable should name a CICS transient data queue. If the MQTRACE variable is not set, diagnostic trace messages are discarded.

Client tracing can introduce significant processing overhead for client applications, and should not be used except in conjunction with IBM service for problem resolution.

MQ_USER_ID
This variable specifies the user ID of the client.

MQ_PASSWORD
This variable specifies the password of the client.

Application programming for clients

This section describes the differences between running applications in an WebSphere MQ client environment and running them in the full WebSphere MQ queue manager environment. It also explains how to build and run applications in the WebSphere MQ client for VSE environment, and how to solve potential problems.

In addition, this section describes the WebSphere MQ client bridge which is unique to the z/VSE environment.

Using the message queue interface (MQI)

The WebSphere MQ client for z/VSE supports the following MQI calls:

MQCONN
MQCONNX
MQDISC
MQOPEN
MQCLOSE
MQPUT
MQPUT1
MQGET
MQINQ
MQSET
MQCMIT
MQBACK

Limiting the size of a message

The maximum message length (MaxMsgLength) attribute of a queue manager is the maximum length of a message that can be handled by that queue manager. The default maximum message length supported depends on the platform you are using.

You can find out the value of MaxMsgLength for a queue manager by using the MQINQ call.

If the MaxMsgLength attribute is changed, no check is made that there are not already queues, and even messages, with a length greater than the new value. After a change to this attribute, applications and channels should be restarted in order to ensure that the change has taken effect. It will then not be possible for any

new messages to be generated that exceed either the queue manager's MaxMsgLength or the queue's MaxMsgLength (unless queue manager segmentation is allowed).

The maximum message length in a channel definition limits the size of a message that you can transmit over a client connection. If an WebSphere MQ application tries to use the MQPUT call or the MQGET call with a message larger than this, an error code is returned to the application.

Coded character set identifiers (CCSID)

The data passed across the MQI from the application to the client stub should be in the local coded character set identifier (CCSID), encoded for the WebSphere MQ client. If the connected queue manager requires the data to be converted, this is done by the client support code.

The client code assumes that the character data crossing the MQI in the client is in the CCSID configured for that machine. If this CCSID is an unsupported CCSID or is not the required CCSID, it can be overridden with the MQCCSID environment variable, for example:

```
ENVAR=('MQCCSID=1047')
```

Set this in the application environment and all MQI data mim89cat is assumed to be in code page 1047.

Note that this does not apply to application data in the message.

If your application is performing multiple PUTs that include WebSphere MQ headers after the message descriptor (MQMD), be aware that the CCSID and encoding fields of the MQMD are overwritten after completion of the first PUT. After the first PUT, these fields contain the value used by the connected queue manager to convert the WebSphere MQ headers. Ensure that your application resets the values to those it requires.

Using MQINQ

Some values queried using MQINQ are modified by the client code.

- CCSID is set to the client CCSID, not that of the queue manager.
- MaxMsgLength is reduced if it is restricted by the channel definition. This is the lower of:
  - The value defined in the queue definition, or
  - The value defined in the channel definition.

For more information, see the *WebSphere MQ Application Programming Guide*.

Using syncpoint coordination

Within WebSphere MQ, one of the roles of the queue manager is syncpoint control within an application. If an application runs on an WebSphere MQ client, it can issue MQCMIT and MQBACK, but the scope of the syncpoint control is limited to the MQI resources.

Applications running in the full queue manager environment on the server can coordinate multiple resources (for example databases) via a transaction monitor. On the server you can use the Transaction Monitor supplied with the Version 5.1 WebSphere MQ products, or another transaction monitor such as CICS. You cannot use a transaction monitor with a client application. The WebSphere MQ verb MQBEGIN is not valid in a client environment.

Using MQCONNX

MQCONNX can be used from a client but only with the following MQCNO options:

MQCNO_NONE

MQCONNX and MQCONN on a client are similar calls, except that MQCONNX allows a client application to specify a channel data (MQCD) structure in the MQCNO structure. This allows the calling client application to specify the definition of the client-connection channel at run time. The actual call issued at the server depends on the server level and the listener configuration.

Building applications for WebSphere MQ clients

The WebSphere MQ client for z/VSE can be used by applications written in Language Environment C/VSE, COBOL/VSE and PLI/VSE.

Building applications for both environments

You can build an WebSphere MQ application for both the full WebSphere MQ environment and the WebSphere MQ client environment without changing your code, provided that:

- It does not need to connect to more than one queue manager concurrently.
- The queue manager name is not prefixed with an asterisk (*) on an MQCONN or MQCONNX call.

Note: The libraries you use at link-edit time determine the environment in which your application must run.

When working in the WebSphere MQ client environment, remember that:

- Each application running in the WebSphere MQ client environment has its own connections to servers. It has one connection to each server it requires, a connection being established with each MQCONN or MQCONNX call the application issues.
- An application sends and gets messages synchronously.
- Message data conversion can be managed by the server.

Triggering in the client environment

Triggering is explained in detail in the *WebSphere MQ Application Programming Guide*.

Messages sent by WebSphere MQ applications running on WebSphere MQ clients contribute to triggering in exactly the same way as any other messages, and they can be used to trigger programs on the server. The trigger monitor and the application to be started must be on the same system.

The default characteristics of the triggered queue are the same as those in the server environment. In particular, if no MQPMO syncpoint control options are specified in a client application putting messages to a triggered queue that is local to a queue manager, the messages are put within a unit of work (if the server queue manager is running on z/OS or z/VSE). If the triggering condition is then met, the trigger message is put on the initiation queue within the same unit of work and cannot be retrieved by the trigger monitor until the unit of work ends. The process that is to be triggered is not started until the unit of work ends.

Linking applications with the WebSphere MQ client

When building applications for the WebSphere MQ client environment, for CICS or batch, you must prelink your application with the WebSphere MQ client for z/VSE MQI object file, MQICVSE.OBJ. For example:

```
// OPTION CATAL
   PHASE YOURPROG,*
   INCLUDE YOURPROG
   INCLUDE MQICVSE
// EXEC EDCPRLK
```

The MQICVSE.OBJ file resides in the WebSphere MQ client for z/VSE client installation sublibrary.

Following the prelink, you must link-edit your application. For example:

```
/*
// EXEC LNKEDT
```

The WebSphere MQ client bridge

The WebSphere MQ client for z/VSE bridge is unique to the VSE environment. The client bridge is managed by a long-running CICS transaction, MQCI. This transaction must be active for the client bridge to be available.

The MQCI transaction can be started in native CICS or via the CICS START command. The MQCI transaction requires a parameter to specify a bridge ID. For example:

```
MQCI mqbisrv1
```

The bridge ID can be 1-8 alpha-numeric characters. Client applications running outside CICS can use the interface by naming the bridge ID via a // SETPARM card (or equivalent), as follows:

```
// SETPARM MQBISRV=mqbisrv1
```

If the bridge transaction (MQCI) is started without an ID parameter, the default name MQBISERV is used. Similarly, if the application does not specify an ID via the //SETPARM card (or equivalent), then the default name MQBISERV is used.

The bridge can be stopped by specifying "X". For example,

```
MQCI X
```

For each client connection, the bridge starts a "partner" transaction (MQCI) that acts as a proxy for the client program. The client installation sublibrary includes file MQCICSD.Z which provides sample CICS System Definitions for the bridge programs and transactions. These must be defined to your CICS system before the bridge can be used.

Building client bridge applications

Client application programs that intend to use the client bridge must be link-edited with the WebSphere MQ Client for z/VSE bridge MQI object file, MQBIBTCH.OBJ. For example:

```
// OPTION CATAL
   PHASE YOURPROG,*
   INCLUDE YOURPROG
   INCLUDE MQBIBTCH
// EXEC LNKEDT
```

Unlike normal z/VSE client applications, programs that use the client bridge MQI (MQBIBTCH), do not require prelink.

The MQBIBTCH.OBJ file resides in the client installation sublibrary. It provides an entry point for each of the support MQI calls listed in "Using the message queue interface (MQI)" on page 634.

WebSphere MQ client bridge security

The WebSphere MQ MQ Client for z/VSE bridge can run in a secure mode. When running in secure mode, the bridge transaction attempts to start proxy transactions as the user issuing the MQCONNX call. In this way, the proxy transaction can run with the same privileges as the user issuing the MQI calls from outside CICS.

The client bridge can be started in secure mode by specifying the SEC=YES parameter, for example:

```
MQCI mqcisrv1 SEC=YES
```

The secure mode requires that the CICS region is started with the SEC=YES SIT parameter, and that the z/VSE system is started with the SEC=YES IPL SYS parameter. If security is not active for both CICS and your z/VSE system, you should not run the Client Bridge in secure mode.

The user that starts the bridge transaction must be a surrogate for client application users. In addition, the user used by the bridge transaction is taken from the // ID card (or equivalent) of the client application environment.

The client bridge and batch interface

The WebSphere MQ client for z/VSE bridge works co-operatively with the WebSphere MQ for z/VSE Batch Interface. Batch applications, link-edited with the Batch Interface MQI (also MQBIBTCH) run client connections if they identify a client bridge by means of // SETPARM MQBISRV.

Similarly, client applications, link-edited with the client bridge MQI (MQBIBTCH) run batch connections if they identify a Batch Interface by means of // SETPARM MQBISRV.

Running applications on WebSphere MQ clients

When an application running in an WebSphere MQ client environment issues an MQCONN or MQCONNX call, the client identifies how it is to make the connection. When an MQCONNX call is issued by an application, the MQI client library searches for the client channel information in the following order:

1. Using the contents of the ClientConnOffset or ClientConnPtr fields of the MQCNO structure (if supplied). These identify the channel data structure (MQCD) to be used as the definition of the client connection channel.
2. If the MQSERVER environment variable is set, the channel it defines is used.

The first of these options (using the ClientConnOffset or ClientConnPtr fields of MQCNO) is supported only by the MQCONNX call. If the application is using MQCONN rather than MQCONNX, the channel information is obtained using the MQSERVER environment variable. If the client fails to find the channel information, the MQCONN or MQCONNX call fails.

The channel name (for the client connection) must match the server-connection channel name defined on the server for the MQCONN or MQCONNX call to succeed.

Using the MQCNO structure

The MQCONNX MQI call is documented in the WebSphere MQ Application Programming Reference manual, along with the MQCNO and MQCD data structures.

The MQCD data structure contains a set of variables that describe the client-connection channel, including the channel name, transport type (which must always be TCP/IP in the z/VSE environment) and the connection name. The connection name identifies the server's hostname or IP address and the port number of the server's MQ listener task.

The MQCNO data structure, which is passed as a parameter to the MQCONNX call, includes an address or an offset for the call to find the MQCD data structure. The MQCONNX call uses this information to establish a connection with the server.

Once connected, the server is available to satisfy MQI requests and respond to the client application with return and reason codes.

Using MQSERVER

Environment variables in z/VSE can be set in two general ways:

- Application program call.
- Language Environment run-time option.

The C run-time call setenv() allows application programs to set environment variables. This call however is not available to COBOL applications. Consequently, the WebSphere MQ client for z/VSE provides function MQSETENV() which allows applications written in COBOL, C or PL/I to set environment variables. The WebSphere MQ client for z/VSE also provides function MQGETENV() for completeness.

Applications written in the C language can alternatively use the #pragma runopts compiler directive to set environment variables using the Language Environment ENVAR run-time option. For example:

```
#pragma runopts (ENVAR("MQSERVER=CLI1.CHAN/TCP/1.11.1.11(1414)"))
```

An application written in PL/I can use the PLIXOPT declaration to specify an ENVAR setting, for example:

```
dcl plixopt char(100) var
    init('ENVAR("MQSERVER=CLI1.CHAN/TCP/1.11.1.11(1414)")')
    static external;
```

The Language Environment ENVAR option recognizes a special environment variable called _CEE_ENVFILE. This special variable can be used to name a file that contains a list of environment variable settings. For example:

```
#pragma runopts (ENVAR("_CEE_ENVFILE=DD:MQVARS.Z"))
```

In this example, the MQVARS.Z files might be created with the following batch job:

```
// JOB LIBRCAT
// EXEC LIBR
ACCESS S=lib.sublib
CATALOG MQVARS.Z  EOD=XX
MQSERVER=CLI1.CHAN/TCP/1.11.1.11(1414)
```

```
MQCCSID=1047
XX
/*
/&
```

The _CEE_ENVFILE variable is not supported in CICS.

Alternatively, environment variables can be set when the application is run by using the ENVAR option as a parameter on the EXEC card. For example:

```
// EXEC YOURPROG,PARM='/ENVAR("MQSERVER=CLI1.CHAN/TCP/1.11.1.11(1414)")'
```

If you are using the WebSphere MQ client for z/VSE bridge to run an application in an environment that does not support Language Environment for z/VSE, the ENVAR option is not available. In such a case, the MQCONNX call must be used.

Solving WebSphere MQ client problems

An application running in the WebSphere MQ client environment receives MQRC_* reason codes in the same way as WebSphere MQ server applications. However, there are additional reason codes for error conditions associated with WebSphere MQ 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 WebSphere MQ client is linked with the correct library file.

WebSphere MQ client fails to make a connection

When the WebSphere MQ client issues an MQCONN or MQCONNX call to a server, socket and port information is exchanged between the WebSphere MQ client and the server. For any exchange of information to take place, there must be a program on the server machine whose role is to 'listen' on the communications line for any activity. This program is called the MQ listener.

If there is no program doing this, or there is one but it is not functioning correctly, the MQCONN or MQCONNX call fails, and the relevant reason code is returned to the WebSphere MQ application.

If the connection is successful, WebSphere MQ protocol messages are exchanged and further checking takes place. During the 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 the *WebSphere MQ Application Programming Reference manual*.

Stopping WebSphere MQ clients

Even though an WebSphere MQ client has stopped, it is still possible for the process at the server to be holding its queues open. The queues is closed when the communications layer detects that the partner has gone.

Error messages with WebSphere MQ clients

When an error occurs with an WebSphere MQ client system, error messages are put into the error files associated with the server, if possible.

In addition, the WebSphere MQ client may attempt to place the error message in an error log. The WebSphere MQ client for z/VSE uses the MQDATA environment variable to identify a target for client log messages.

The MQDATA environment variable, if used in batch, should name a DLBL of a VSAM managed SAM file, or an ESDS. Message lengths are variable up to 100 characters. If used in CICS, the MQDATA variable should name a CICS transient data queue. If the MQDATA variable is not set, client log messages are discarded.

Tracing WebSphere MQ clients

The WebSphere MQ client for z/VSE uses the MQTRACE environment variable to identify a target for diagnostic trace messages.

The MQTRACE environment variable, if used in batch, should name a DLBL of a VSAM managed SAM file, or an ESDS. Message lengths are variable up to 80 characters. If used in CICS, the MQTRACE variable should name a CICS transient data queue. If the MQTRACE variable is not set, diagnostic trace messages are discarded.

Client tracing can introduce significant processing overhead for client applications, and should not be used except in conjunction with IBM service for problem resolution.

Example client trace: Figure 83 shows an extract from an WebSphere MQ for z/VSE client trace.

```
WebSphere MQ Trace started at 04/19/05 16:00:28
<- xcsInitialize (rc = OK)
-> MQCONN
--> rrxOpenChannelDef
---> xcsGetMem
<--- xcsGetMem (rc = OK)
<-- rrxOpenChannelDef (rc = OK)
--> rrxGetFirstChannelDef
<-- rrxGetFirstChannelDef (rc = OK)
--> rriInitSess
---> xcsGetMem
<--- xcsGetMem (rc = OK)
.
.
.
---> rriTermExits
<--- rriTermExits (rc = OK)
---> rriDeleteStatusEntry
----> xcsFreeMem
<---- xcsFreeMem (rc = OK)
<--- rriDeleteStatusEntry (rc = OK)
---> xcsFreeMem
<--- xcsFreeMem (rc = OK)
<-- rriFreeSess (rc = OK)
<- MQDISC
```

Figure 83. Extract from WebSphere MQ for z/VSE client trace

Chapter 11. Secure Sockets Layer services

Secure Sockets Layer (SSL) is a communications protocol that provides secure communications over an open communications network (for example, the Internet). The SSL protocol is a layered protocol that is intended to be used on top of a reliable transport, such as Transmission Control Protocol (TCP/IP). SSL provides data privacy and integrity as well as server and client authentication based on public key certificates. Once an SSL connection is established between a client and server, data communications between client and server are transparent to the encryption and integrity added by the SSL protocol.

WebSphere MQ for z/VSE incorporates SSL services between itself and remote queue managers and MQ clients that also incorporate SSL services. From an SSL perspective, in every case, the initiating application is considered the client and the remote application accepting the connection, the server. From an WebSphere MQ perspective, the client is a remote sender Message Channel Agent (MCA) or an WebSphere MQ client, and the server is the receiver MCA.

WebSphere MQ activates SSL services on a per channel basis. This is possible through the channel definition. Each channel can identify whether or not SSL services are required when a connection is made or accepted, to or from a remote system or client program.

There are a number of steps involved in establishing an SSL enabled and active channel. These include:
- Installing the SSL feature.
- Configuring the queue manager for SSL.
- Configuring a channel for SSL.
- Activating SSL services.

Each of these is described in some detail below.

Installing the SSL feature

Before SSL channels can be established by WebSphere MQ, the SSL feature must be installed and available under the z/VSE environment.

SSL for z/VSE is an optional product that is integrated into TCP/IP for z/VSE. As an optional product, it requires a special product code to activate its features. Full details for installation should be found in SSL for z/VSE documentation.

Part of the installation process, of immediate relevance to WebSphere MQ, is the creation of the SSL key-ring sublibrary. The SSL key-ring sublibrary contains private key members (.PRVK files) and X.509 certificate files (.CERT files).

If SSL enabled channels are required, WebSphere MQ for z/VSE must be configured with the name of the SSL key-ring sublibrary, and the name of the queue manager's private key and certificate files. This configuration is part of the queue manager definition.

Configuring the queue manager for SSL

Once the SSL feature has been installed and is available under z/VSE, the WebSphere MQ queue manager can be configured to identify the SSL key-ring sublibrary and private key and certificate files.

Access SSL configuration for the queue manager from the Global System Definition maintenance screen (MQMT option 1.1), using PF9. Pressing PF9 displays the queue manager "Communication Settings" (Figure 84).

```
2012/11/14        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
15:33:21                   Global System Definition                  CIC1
MQWMSYS                   Queue Manager Information                  A004
Queue Manager . . . . . .: PTHVSEC
Description Line 1. . . . .: TEST
Description Line 2. . . . .:
                          System Values
Maximum Connection Handles.: 00000100       System Wait Interval : 00000030
Maximum Concurrent Queues .: 00000100       Max. Recovery Tasks  : 0000
Allow TDQ Write on Errors  : Y   CSMT       Local Code Page  . . : 01047
Allow Internal Dump  . . . : Y              Subsystem id . . . . : MQV1
                                            Channel Auth Enabled : N
                      Queue Maximum Values
Maximum Q Depth . . . . . .: 00100000       Maximum Global Locks.: 00001000
Maximum Message Size. . . .: 00409600       Maximum Local Locks .: 00001000
Maximum Single Q Access . .: 00000100       Max Properties Length: 00004094
                      Global QUEUE /File Names
Configuration File. : MQFCNFG
LOG Queue Name. . . : SYSTEM.LOG
Dead Letter Name. . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name. : SYSTEM.MONITOR
Requested record displayed.
PF2=Return PF3=Quit PF4/ENTER=Refresh    PF6=Update
                    PF9=Communications  PF10=Log    PF11=Events  PF12=Exits
```

Figure 84. Queue manager communication settings

Queue manager communication settings are divided into five categories:
- TCP/IP settings
- SSL parameters
- PCF parameters
- Batch interface settings
- Channel auto-definition

Of immediate relevance to SSL enabled channels are the TCP/IP settings and the SSL parameters.

TCP/IP settings

TCP/IP listener port

The TCP/IP listener port represents the IP port number on which the MQ Listener program will accept remote connection requests. The MQ Listener is a long running transaction, MQTL.

Caution should be taken not to use a port number that is already in use by another application or subsystem. The default value for WebSphere MQ is 1414.

Licensed clients

The number of licensed clients represents the maximum number of concurrent client connections that will be accepted by the queue manager. The number allowed is determined by your WebSphere MQ for z/VSE license agreement.

You should check your license agreement for the number of concurrent client connections permitted. If a restriction is not applicable, you can set the licensed clients value to zero.

SSL parameters

Key-ring sublibrary

The key-ring sublibrary identifies the SSL key-ring sublibrary, identified and generated during SSL for z/VSE installation. The key- ring sublibrary contains private key and X.509 certificate files. The value entered should be a valid z/VSE sublibrary name.

If a key-ring sublibrary is specified, WebSphere MQ will perform SSL initialization during system startup, even if there are no SSL enabled channels. If SSL is not installed, this field should be left blank.

Key-ring member

The key-ring member identifies the SSL key-ring sublibrary member name of the private key and certificate files that will be used by WebSphere MQ enabled channels. This must be a valid z/VSE sublibrary member name.

It should be noted that WebSphere MQ for z/VSE uses the same private key and certificate for all SSL enabled channels. It is not possible to identify a different certificate on a per channel basis. Consequently, the key-ring member name should identify a private key and certificate files appropriate for all SSL enabled channels.

SSL reset count

During an SSL handshake a secret key is generated to encrypt data between the SSL client and SSL 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.

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.

Once the secret key has been renegotiated, the previous secret key can no longer be used to decrypt data encrypted with the new secret key.

The SSL reset count specifies the total number of unencrypted bytes that are sent and received within an SSL conversation before the secret key is renegotiated. The number of bytes includes control information sent by the message channel agent.

Note: When a value greater than 0 but less than 32000 is specified, the value is ignored, and 32000 is used.

It should further be noted that an SSL renegotiation is resource intensive, and can place significant overhead on SSL-enabled channel processing time. Consequently, when using SSL key reset, it is recommended that you use a value as high as possible but does not compromising the channel's security.

Configuring a channel for SSL

Channels can be enabled for SSL. In the case of Sender, Server, Receiver and Requester channels, SSL enablement assumes that the partner channel definition (on a remote WMQ system) is also configured for SSL. For client channels, WMQ documentation for the relevant client system should be reviewed to determine how to enable a client for SSL.

Note that reference to Sender channels in this chapter refers generically to sender and server (and requester channels during channel initialization), which send messages to remote systems. Reference to Receiver channels refers to both receiver and requester channels, which receive messages from remote systems.

Configure the SSL parameters for a channel using the “Channel SSL Parameters” screen. To get to this screen, press PF10 at the “Maintain Channel Record” screen (MQMT option 1.3)

```
11/28/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
12:44:28                  Channel SSL Parameters                     CIC1
MQMMCHN                                                              A003

     Channel Name: VSE1.TCP.NT1           Type: S

     SSL Cipher Specification. : 02       (2 character code)
     SSL Client Authentication : R        (Required or Optional)

     SSL Peer Attributes:
     > O=IBM,OU="Australian Programming Centre",C=Australia,ST=WA,L=Per <
     > th,CN=www.ibm.com                                               <
     >                                                                 <
     >                                                                 <









SSL channel parameters displayed.
    PF2 = Return    PF3 = Quit     PF4 = Read      PF6 = Update
```

Figure 85. SSL parameters for a channel

SSL parameters are identical, regardless of channel type. Consequently, the above screen is applicable to sender, receiver and client channels.

Secure Sockets Layer services are only available for TCP/IP channels. Consequently, SSL parameters for SNA LU 6.2 channels are ignored.

SSL channel parameters

Channel name
The name of the channel for which the SSL parameters apply. This is a display field only.

Type
The type of the channel for which the SSL parameters apply. This is a display field only.

SSL Cipher Specification
The SSL Cipher Specification is a two-character code that identifies an SSL version 3 cipher specification supported by the SSL for z/VSE feature. For example:

Table 13. Supported SSL cipher specifications

Cipher	Description
01	NULL MD5
02	NULL SHA
08	DES40 SHA for Export
09	DES SHA for U.S.
0A	Triple DES SHA for U.S.
62	RSA_EXPORT1024_DESCBC_SHA
2F	RSA AES128 CBC SHA
35	RSA AES256 CBC SHA

The code selected must be supported on the remote system. In the case of a sender channel, WebSphere MQ for z/VSE will establish an SSL enabled channel with a remote MQ system only if the remote system accepts the specified code. For receiver channels, the remote system will identified the desired code. If the local SSL feature supports the designated code, channel initialization will proceed. Otherwise the channel is terminated with an error.

It should be noted that this parameter determines whether or not the channel is SSL enabled. If this field is blank, the other SSL parameters are ignored, and the channel operates without SSL services. Any non-blank value means the channel is SSL enabled, and the other SSL parameters are used during channel initialization.

SSL Client Authentication
The SSL Client Authentication field can be set to 'R' for required, or 'O' for optional. If client authentication is required, WebSphere MQ checks that a certificate was sent from the remote system during SSL initial negotiation. If not, the channel is terminated with a error.

Since a certificate is always sent by the receiver (or SSL server) to the sender (or SSL client), this field is only meaningful to receiver and client channels. However, the WebSphere MQ for z/VSE Receiver MCA, which acts as the SSL server, requires that SSL clients send a certificate during SSL negotiation. Consequently, under WebSphere MQ for z/VSE, a client certificate will always be received during SSL negotiation. If not, the channel is terminated with an initialization error.

The SSL client authentication field, therefore, exists for compatibility with other WebSphere MQ systems and possible future expansion. To make this apparent, this field should be set to 'R'.

SSL Peer Attributes

The SSL Peer Attributes parameter allows a channel to verify that the partner's certificate contains certain identifiable characteristics. If the partner's certificate does not contain these characteristics, the channel is terminated with an error.

Identifiable characteristic types include:

Table 14. SSL Peer Attribute types

Type	Description
CN	Common name
L	Locality
ST	State or province
C	Country
O	Organization
OU	Organizational unit
SERIAL	Serial number

The characteristic types pertain to the Subject attributes of the X.509 certificate, except for serial number, which pertains to the Serial Number of the certificate.

The SSL Peer Attributes field takes this form:

type=value, *type=value*,...Where *type* is one of the characteristic types listed in Table 14, the equals sign (=) is constant, and *value* identifies an expected value relevant to the characteristic type specified. For multiple attributes, the comma (,) is also required and constant. For example:

```
O=IBM,C=US
```

In this example, the remote partner's certificate must have a Subject Organization of "IBM", and a Subject Country of "US".

The SSL Peer Attributes parameter will also accept wildcards (*). Each value can have only one wildcard. Wildcards cannot be imbedded in a value. For example:

```
O=IBM,OU=LAB*,C=UK
```

In this example, the channel will accept remote certificates with a Subject Organization of "IBM", a Subject Country of "UK" and any Subject Organizational Unit beginning with "LAB".

The SSL Peer Attributes parameter will also accept imbedded spaces. These must be enclosed in double quotes ("). For example:

```
O="IBM GSA"
```

Double quotes are optional for values that do not contain imbedded blanks.

If the SSL Peer Attribute parameter is set and the remote certificate does not match its stipulations, the channel is terminated with an error. If the SSL Peer Attributes is not set (it is left blank), the remote certificate's identifying attributes are not examined. In other words, a blank parameter means the channel can be activated by any valid certificate.

Activating SSL services

When a channel is configured for SSL, WebSphere MQ automatically activates SSL services when the channel is established.

For Sender channels, the Sender Message Channel Agent (MCA), trigger program MQPSEND, examines the relevant channel definition to check if the SSL Cipher Specification field has been set. If so, WebSphere MQ considers the channel "SSL enabled" and will attempt to establish an SSL connection with the remote queue manager using SSL services.

For Receiver and Client channels, the Receiver MCA does not, on invocation, have details of the channel. Consequently, it cannot examine the SSL Cipher Specification parameter. Instead, the Receiver MCA examines the initial dataflow from the remote queue manager. If it is identifiable as an SSL exchange, the Receiver MCA will attempt to secure the connection using SSL services. If successful, subsequent dataflow will identify the appropriate channel.

It is at this point that WebSphere MQ will verify that the correct SSL Cipher Specification, SSL Client Authentication and SSL Peer Attributes have been received or negotiated. If not, the channel is immediately terminated with an error.

This means that for all SSL enabled channels, the dataflow, from start to finish, is under SSL control. This included WebSphere MQ' initial channel negotiation which contains channel, queue manager and target queue information.

The SSL Peer Attributes parameter adds further protection by allowing a channel to reject any connection that uses a certificate that does not meet the stipulations of the parameter.

Chapter 12. Security

This chapter describes the features of security control in WebSphere MQ for z/VSE and how you can implement and manage this control.

Note: Examples in this chapter use CA-Top Secret as an external security manager (ESM). If you are using a different ESM, you should modify the techniques described. With z/VSE 4.3, the Basic Security Manager (BSM) now has the MQ classes required to support WebSphere MQ for z/VSE security.

Where profile names are shown, replace the subsystem identifier (SSID) in the profile name with the name of the WMQ subsystem you are using. The subsystem name for your queue manager is a 4-character identifier determined during installation. See "Installing security" on page 19 for information about the SSID.

For profile names, WebSphere MQ for z/VSE uses the full queue manager name instead of the subsystem identifier if the SSID is blank (spaces). When using the Basic Security Manager, it is recommended that the SSID is used in place of the queue manager name so that the profile name fits the maximum length accepted by the Basic Security Manager.

Why you need to protect WebSphere MQ resources

Because WebSphere MQ handles the transfer of information that is potentially valuable, you need the safeguard of a security system. This ensures that the resources that WebSphere MQ owns and manages are protected from unauthorized access, which could lead to the loss or disclosure of the information. In a secure system, it is essential that none of the following are accessed or changed by any unauthorized user or process:

- Connections to WebSphere MQ.
- WebSphere MQ objects such as queue managers and queues.
- WebSphere MQ transmission links.
- WebSphere MQ system control commands.
- WebSphere MQ messages.
- Context information associated with messages.

To provide the necessary security, WebSphere MQ uses the z/VSE system authorization facility (SAF) to route authorization requests to an ESM, for example, CA-Top Secret. With z/VSE 4.3, the Basic Security Manager (BSM) now includes the MQ classes and so may be used in place of an ESM.

The decision to allow access to an object is made by the ESM or BSM, and WebSphere MQ follows that decision. If the ESM or BSM cannot make a decision, WebSphere MQ prevents access to the object by default. However, by default, if the CICS system running WebSphere MQ is configured without security, WebSphere MQ will not restrict access to its resources.

Implementing WebSphere MQ security

It is easier to set up and administer your security if first you decide on a set of naming conventions for your WebSphere MQ objects.

To implement a security strategy for your WebSphere MQ subsystem, you must decide:

- How security is to be used and implemented.
- Who is going to use the WebSphere MQ system and resources.

To use the CA-Top Secret examples, as shown in this manual, you must be a suitably authorized user, for example, the MSCA user. You can enter the commands either from CICS or via a batch job, using the TSS transaction or the TSSCMNDB program, respectively.

In the same way, if you use the Basic Security Manager you must be a suitably authorized user to use the Interactive User Interface Security Maintenance panel or the BSTADMIN batch program.

Resources you can protect

When WebSphere MQ starts, or when it is instructed by an operator command, WebSphere MQ determines which resources you want to protect. You can control which security checks are performed for each individual queue manager. For example, you could implement a number of security checks on a production queue manager, but none on a test queue manager.

Objects protected by WebSphere MQ for z/VSE include:

- Connections
- Queues
- Namelists
- Messages
- Commands
- Command resources

This chapter also explains how you might protect WebSphere MQ datasets, specifically, the VSAM files used by WebSphere MQ.

Connection security

Connection security checking occurs either when an application program tries to connect to a queue manager by issuing an MQCONN request, or when WebSphere MQ itself issues a connection request. You can turn connection security checking off for a particular WebSphere MQ subsystem, but if you do, any user can connect to that subsystem.

WebSphere MQ itself issues a connection request when it attempts to log messages to the system log. The logging mechanism writes messages to a transient data queue (MQER) that triggers a transaction to write the message to the system log queue. This transaction (MQER) runs as the CICS default user, or a user specified in the DCT entry for the transient data queue. At installation, you must decide whether to use the default user or a specific user to connect and write messages to the system log.

Similarly, the message expiry feature of WebSphere MQ uses a transient data queue. The message expiry transient data queue (MQXP) is defined in the CICS Destination Control Table (DCT) to fire a trigger transaction (also MQXP) when the data queue contains at least one entry. The MQXP transaction is responsible for clearing expired messages from application queues, and for generating expiry report message when they are required.

Like the MQER transaction, the MQXP transaction runs with the authority of the CICS default user unless the DCT definition includes the USERID parameter. See “Changing the MQXP TDQ definition” on page 22. Since the MQXP transaction may need to place a report message on any application queue, the user that runs the transaction must have suitable authority. The user will at least require connect authority. This is the case for the MQAC transient data queue and MQAC transaction (used for accounting and statistics), and the MQIE transient data queue and MQIE transaction (used to write event messages to the event queues).

Queue and message security

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on which open options are specified when the queue is opened.

A security check is performed when the queue manager object is opened. In this situation, the queue manager is protected in the same way as a queue object, that is, a user must have permission to access `ssid.qmname`, where `qmname` is the name of your queue manager.

Queue security controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (via the MQOO_BROWSE option), but not to remove messages from the queue (via one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

Namelist security

Similar to queue security, namelist security checking is carried out when a user opens a namelist.

If you turn checking for namelists off, any user can open any namelist.

When namelist security is active, permissions are checked when an application issues an MQOPEN call. Namelists can only be opened for inquiry. For example MQOO_INQUIRE, so the only applicable permission specifically required to open a namelist is read access.

Command security

Command security involves protecting the creation, deletion, and modification of WebSphere MQ objects. Commands that affect WebSphere MQ objects can be issued in three ways:

- Master terminal commands
- PCF commands
- WebSphere MQ commands

Master terminal commands are those entered interactively in native CICS, or via the CICS Web Support feature from a web browser. Master terminal commands are generally selected from the primary options menu of the master terminal transaction (MQMT).

PCF commands are processed as data messages by the WebSphere MQ command server which is a long-running CICS transaction (MQCS). The command server reads PCF messages from system command queue. The name of this queue is

configurable as a communication parameter of the queue manager's global system definition. Message read from the system command queue are expected to be PCF commands which are parsed and processed by the WebSphere MQ command processor (transaction MQCX). For more information about PCF commands, refer to Chapter 8, “Programmable system management,” on page 221.

WebSphere MQ commands are verb-based text messages processed by the WebSphere MQ MQSC batch utility program (MQPMQSC). The MQSC utility program converts SYSIPT verb-based command text into PCF Escape messages and places them on the system command queue. Responses to these commands are placed on the system reply queue (a communication parameter of the global system definition). The MQSC utility processes MQSC reply messages and generates SYSLST text output. For more information about WebSphere MQ commands and the command utility, refer to Chapter 9, “WebSphere MQ commands,” on page 507.

Command security involves authorization by command type. For example, a user might be authorized to issue 'DISPLAY' commands, but not 'ALTER' commands. Similarly, a user may be authorized to perform the display options but not the maintenance options under MQMT option 1 (Configuration).

The userid on the batch job // ID must have UPDATE access to the system admin command and reply queues.

Command resource security

Command resource security involves protecting WebSphere MQ objects by name, and works in conjunction with command security. For example, a user may be authorized to issue 'DISPLAY' commands, but may be restricted to displaying objects with a certain prefix.

Consequently, for a user to display the details of a specific queue, for example, that user would need command authorization to 'DISPLAY' and command resource authorization to display details of the specific queue.

Dataset security

WebSphere MQ for z/VSE queues are implemented as VSAM KSDS datasets, and WebSphere MQ configuration is also stored in VSAM datasets. Therefore, it is important that these datasets are protected against unauthorized access under z/VSE generally.

Check your ESM or Basic Security Manager documentation for specific details on protecting datasets. WebSphere MQ assumes that users with authorization to specific queues, with specific access permissions, are also authorized to the datasets that contain queue data, with the same permissions. This assumption relies on the security administrator correlating the correct permissions for queues and datasets.

See Appendix H, “Security implementation,” on page 1055 for more details on how to protect your datasets.

Using security classes and resources

CA-Top Secret and the Basic Security Manager classes are used to hold the resources required for WebSphere MQ security checking. Each class holds one or more resources used at some point in the checking sequence.

Table 15. Classes used by WebSphere MQ

Member class	Description
MQADMIN	Used mainly for holding resources for administration-type functions. For example, profiles for WebSphere MQ security switches.
MQCONN	Profiles used for connection security.
MQNLIST	Profiles used in namelist resource security.
MQQUEUE	Profiles used in queue resource security.
MQCMDS	Profiles for command security.
MXTOPIC	Profiles for topic security.

Depending on your External Security Manager, these classes may be predefined. For CA-Top Secret, these are predefined Prefixed resources. To activate such resources, ensure that the following setting exists in your CA-Top Secret parameter file:

```
FACILITY(CICSPROD=RES)
```

The Basic Security Manager has the SURROGAT class in addition to the above classes. Use the batch BSTADMIN to set these classes active. For example JCL, see Appendix H, “Security implementation,” on page 1055.

Note:

1. CICSPROD should be replaced by the facility you are using for your WebSphere MQ CICS region, if it is different.
2. After you change the parameter file, you need to restart your ESM.
3. If you are using the Basic Security Manager, then either use the batch BSTADMIN REFRESH command or the "BSM Security Rebuild" option of the Security Maintenance panel, after setting classes active.

Resources

All resources (BSM profile names) used by WebSphere MQ are prefixed with the name of the subsystem that they are to be used by. For example, if queue manager with SSID MQV1 has a queue called QUEUE_FOR_LOST_CARD_LIST, the appropriate profile would be defined to the ESM or BSM in class MQQUEUE as:

MQV1.QUEUE_FOR_LOST_CARD_LIST

This means that different WebSphere MQ subsystems sharing the same ESM database or BSM control file can have different security options. The subsystem identifier for the resource cannot be generic.

If your MQ object name contains lowercase characters, or the characters %, /, or _, then specify the profile name within quotes. For example:

```
'MQV1,QUEUE_FOR_LOST_CARD_LIST'
```

Switch resources

To control the security checking performed by WebSphere MQ, you must define switch profiles. A switch profile is a normal resource that has a special meaning to WebSphere MQ. If you do not want to control security checking, that is, allow WebSphere MQ to check authority for all WebSphere MQ resources, you do not need to define switch profiles.

Each switch profile that WebSphere MQ detects turns off the checking for that type of resource. Switch profiles are activated during startup of the queue manager. If you change the switch profiles while the queue manager is running, the changes will not be recognized until WebSphere MQ is stopped and the WebSphere MQ environment is re-established by the MQSE transaction.

The switch resources must always be defined in the MQADMIN class. The following table shows the valid switch profiles and the security type they control.

Note: In the descriptions that follow, the part of each resource name shown in upper case must be entered exactly as shown. The lower case 'ssid' part must be replaced by the subsystem identifier for the WebSphere MQ subsystem you are setting up.

Table 16. Switch Resources

Switch Resource Name	Description
ssid.NO.SUBSYS.SECURITY	Subsystem security
ssid.NO.CONNECT.CHECKS	Connection security
ssid.NO.NLIST.CHECKS	Namelist security
ssid.NO.QUEUE.CHECKS	Queue security
ssid.NO.CMD.CHECKS	Command security
ssid.NO.CMD.RESC.CHECKS	Command resource security
ssid.NO.TOPIC.CHECKS	Topic security

If you intended to use the security switches, you can create them and grant access as follows:

```
TSS ADD(mqowner) MQADMIN(ssid.NO.CONNECT.CHECKS)
TSS PER(mqstart) MQADMIN(ssid.NO.CONNECT.CHECKS) ACC(READ)
```

For Basic Security Manager (BSM), the BSTADMIN commands are:

```
ADD     MQADMIN   ssid.NO.CONNECT.CHECKS   UACC(NONE)
PERMIT  MQADMIN   ssid.NO.CONNECT.CHECKS   ID(mqstart) ACCESS(READ)
```

In this example, the resource is owned by user `mqowner`, and the `mqstart` user is granted read access to the resource. Note that access to security switch resources is only relevant to the WebSphere MQ for z/VSE startup user (that is, the user who starts WebSphere MQ for z/VSE using MQSE, MQIT, or MQMT).

In the preceding example, security checks for connecting to the WebSphere MQ for z/VSE queue manager would be disabled.

How switches work

WebSphere MQ maintains an internal set of switches, which is associated with each of the switch resources shown in Table 16. When a security switch is set on,

the security checks associated with the switch are performed. When a security switch is set off, the security checks associated with that switch are bypassed.

When a queue manager is started, first it checks the status of the resource switches. The queue manager sets its subsystem security switch off only if the switch resources exist and are readable by the user associated with WebSphere MQ startup. In all other situations, the switches are set on. Note that switches are only applicable when WebSphere MQ is installed with security active.

If the ssid.NO.SUBSYS.SECURITY resource is detected during startup, connection, queue, and message security is bypassed, regardless of other switch settings. This means it is possible to completely disable WebSphere MQ object security by creating the NO.SUBSYS.SECURITY resource, making it readable to the startup user, and re-establishing the WebSphere MQ environment by using the MQSE transaction and then initializing WebSphere MQ using the MQIT transaction.

Take care with generic resources. Some ESMs automatically grant access to resources if the prefix of the resource is owned, or accessible to, a user. For example, if the resource ssid is created and owned by user MQM, and that resource is generic, some ESMs may automatically grant read access to ssid.* to user MQM. The result is that when WebSphere MQ is started up by user MQM, WebSphere MQ will assume all of the switches exist, and all object security will be disabled.

Protecting WebSphere MQ resources

As well as optionally defining switch resources, ESM or BSM resources must be defined to protect the WebSphere MQ objects.

If you do not have a resource profile defined for a particular security check and a user issues a request that would involve making that check, WebSphere MQ denies access.

You do not need to define profiles for security types relating to any security switch profiles that you have deactivated.

Resource definitions for connection security

If connection security is active, you must define profiles in the MQCONN class, and permit the necessary groups or user IDs access to those profiles, so that they can connect to WebSphere MQ subsystems.

To enable a connection to be made, you must grant users READ access to the appropriate profile.

Resource names for checking connections to WebSphere MQ for z/VSE take the form:

ssid.CICS

This applies to CICS applications, batch programs using the batch interface, and remote clients. This is because all connections to WebSphere MQ for z/VSE are effectively maintained within CICS.

For example, to grant user JOHNS connection authority to queue manager VSE.QM1 with SSID MQV1, you must first define the resource and grant ownership:

```
TSS ADD(MQOWNER) MQCONN(MQV1.CICS)
```

You can then grant connection authority as follows:

```
TSS PER(JOHNS) MQCONN(MQV1.CICS) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
ADD    MQCONN  MQV1.CICS  UACC(NONE)
PERMIT MQCONN  MQV1.CICS  ID(JOHNS)  ACC(READ)
```

Depending on your ESM, the owner of a resource may by default have full authority. This would mean that user MQOWNER, in this example, would automatically be granted connection authority to queue manager MQV1.

Batch connections

Security for batch connections is a special case. Batch programs connect to WebSphere MQ for z/VSE running under CICS via the WebSphere MQ for z/VSE batch interface.

Programs executed from a batch partition should use the // ID statement to identify their user and password. Security for batch programs should be established to verify the user and password identified on the // ID card.

A sample batch job might appear as follows:

```
// JOB MQBATCH
// ID USER=JOHNS,PWD=JOHNSPWD
// EXEC MYMQPROG
/*
/&
```

The WebSphere MQ for z/VSE batch interface uses the user name identified in the // ID card and passes it to an interface transaction running under CICS. The interface transaction must be started by, and running as, a user identified to your ESM as a SURROGATE for the user identified on the // ID card.

To identify a user as a surrogate for another, you can use a command similar to:

```
TSS ADD(MQBATCH) SURROGAT(JOHNS)
```

where MQBATCH is the user that starts the batch interface transaction (MQBI) in CICS.

For BSM, the BSTADMIN commands are:

```
ADD    SURROGAT  JOHNS.DFHSTART  UACC(NONE)
PERMIT SURROGAT  JOHNS.DFHSTART  ID(MQBATCH) ACC(READ)
```

When the WebSphere MQ batch program attempts to connect to the queue manager, a check for the surrogate rights of the interface user is issued. If this is successful, a partner transaction (MQBX) is started as the user identified on the // ID card. Therefore, the user identified on the // ID card should be known to CICS.

Once the partner transaction is started, it functions on behalf of the WebSphere MQ batch program. This means that all MQI calls are executed as the user identified on the // ID card. For connection security, this user must be granted READ access to the ssid.CICS resource.

Client connections

Security for client connections is also a special case. For client connections, the client program runs on a remote system. Security for the execution of such programs remains the responsibility of the remote system.

For client programs, the WebSphere MQ for z/VSE server program effectively performs WebSphere MQ API requests on behalf of the client program. The server program runs under CICS and is executed as the WebSphere MQ for z/VSE startup user. The startup user is the user who starts WebSphere MQ for z/VSE using the MQSE, MQIT or MQMT transactions.

The WebSphere MQ for z/VSE server program identifies the client user when the client connection is initiated with the MQCONN call. For authentication, the environment of the client program must include the MQ_USER_ID and MQ_PASSWORD environment variables. The values of these variables are passed to the WebSphere MQ for z/VSE server program when the connection begins. These variables should contain a valid user ID and password, respectively, that are known to the z/VSE ESM or z/VSE BSM.

The WebSphere MQ for z/VSE server program, having identified and verified the client user and password, then performs all security checks for that user, not the WebSphere MQ for z/VSE startup user.

This means that the client user must have the appropriate access to the required ESM resources. This is the same access that would be required for a normal CICS transaction user.

For example, for a client program that identifies itself as user JANED, and intends to connect to WebSphere MQ for z/VSE and browse queue EMPLOYEE.DETAILS on z/VSE queue manager VSE.QM1 with SSID MQV1, you would need to define and grant access to the following resources:

```
TSS PER(JANED) MQCONN(MQV1.CICS) ACC(READ)
TSS PER(JANED) MQQUEUE(MQV1.EMPLOYEE.DETAILS) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
PERMIT MQCONN   MQV1.CICS                ID(JANED) ACC(READ)
PERMIT MQQUEUE  MQV1.EMPLOYEE.DETAILS    ID(JANED) ACC(READ)
```

Because authentication is possible only for client programs that identify themselves using the MQ_USER_ID and MQ_PASSWORD environment variables, WebSphere MQ for z/VSE security for client programs is possible only for remote systems that support this protocol.

Another consideration, which may affect Java program clients, is access permission to the queue manager object. Some existing WebSphere MQ Java classes open the queue manager object when they establish an initial connection. This means that users using WebSphere MQ Java classes should be granted READ access to the WebSphere MQ queue manager object.

For example:

```
TSS PER(cliuser) MQQUEUE(ssid.ssid) ACC(READ)
```

For BSM, the BSTADMIN command is:

```
        PERMIT MQQUEUE  ssid.ssid   ID(cliuser)   ACC(READ)
```

Resource definitions for queue security

If queue security is active, you must define resources in the MQQUEUE class, and permit the necessary groups or user IDs access to these resources, so that they can issue WebSphere MQ API requests that use queues.

Resource names for queue security take the form:

ssid.*queuename*

where *queuename* is the name of the queue being opened, as specified in the object descriptor on the MQOPEN or MQPUT1 call. It may also be the name of the queue manager.

The ESM/BSM access required to open a queue depends on the MQOPEN or MQPUT1 options specified. If more than one of the MQOO_* options is coded, the queue security check is performed for the highest ESM/BSM authority required.

Table 17. Access levels for queue security

MQOPEN or MQPUT1 option	**ESM access level required to access ssid.queuename**
MQOO_BROWSE	READ
MQOO_INQUIRE	READ
MQOO_INPUT_*	UPDATE
MQOO_OUTPUT or MQPUT1	UPDATE
MQOO_SET	ALTER

For example, to grant user JOHNS authority to browse queue PAY.LIST on queue manager VSE.QM1 with SSID MQV1:

```
TSS ADD(MQOWNER) MQQUEUE(MQV1.PAY.LIST)
TSS PER(JOHNS) MQQUEUE(MQV1.PAY.LIST) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
ADD    MQQUEUE  MQV1.PAY.LIST     UACC(NONE)
PERMIT MQQUEUE  MQV1.PAY.LIST     ID(JOHNS)   ACC(READ)
```

Alternatively, to grant user JOHNS authority to get and put messages to queue PAY.LIST on SSID MQV1:

```
TSS PER(JOHNS) MQQUEUE(MQV1.PAY.LIST) ACC(READ,UPDATE)
```

For BSM, the BSTADMIN command is:

```
PERMIT MQQUEUE  MQV1.PAY.LIST     ID(JOHNS)   ACC(UPDATE)
```

Note that the resource only needs to be created, and ownership applied, once. Therefore, the TSS ADD command is issued only once for each queue resource defined to class MQQUEUE. In BSM, the ADD command is also only issued once for each resource defined to class MQQUEUE.

Considerations for alias queues

When you issue an MQOPEN or MQPUT1 call for an alias queue, WebSphere MQ makes a resource check against the queue name specified in the object descriptor (MQOD) on the call. It does not check whether the user is allowed access to the target queue name.

For example, an alias queue called PAYROLL.REQUEST resolves to a target queue of PAY.REQUEST. If queue security is active, a user only needs authorization to access the queue PAYROLL.REQUEST. There is no check whether that user is authorized to access the queue PAY.REQUEST.

Using alias queues with MQGET and MQPUT

The range of MQI calls available in one access level can cause a problem if you want to restrict access to a queue to allow only the MQPUT call, or only the MQGET call. You can protect a queue by defining two aliases that resolve to that queue:

- One that enables applications to get message from the queue.
- One that enables applications to put messages on the queue.

The following text is an example of defining your queue to WebSphere MQ (these definitions are based on OS/2 formats, and you should use the WebSphere MQ for z/VSE Master Terminal transaction to create appropriate definitions):

```
DEFINE QLOCAL(USE_ALIAS_TO_ACCESS) GET(ENABLED)
       PUT(ENABLED)

DEFINE QALIAS(USE_FOR_GETS) GET(ENABLED)
       PUT(DISABLED) TARGQ(USE_ALIAS_TO_ACCESS)

DEFINE QALIAS(USE_FOR_PUTS) GET(DISABLED)
       PUT(ENABLED) TARGQ(USE_ALIAS_TO_ACCESS)
```

You must also make the following ESM definitions:

```
TSS ADD(MQOWNER) MQQUEUE(ssid.USE_ALIAS_TO_ACCESS)
TSS ADD(MQOWNER) MQQUEUE(ssid.USE_FOR_GETS)
TSS ADD(MQOWNER) MQQUEUE(ssid.USE_FOR_PUTS)
```

For BSM, the BSTADMIN commands are:

```
ADD    MQQUEUE  ssid.USE_ALIAS_TO_ACCESS UACC(NONE)
ADD    MQQUEUE  ssid.USE_FOR_GETS        UACC(NONE)
ADD    MQQUEUE  ssid.USE_FOR_PUTS        UACC(NONE)
```

Then, you must ensure that no users have access to the queue ssid.USE_ALIAS_TO_ACCESS, and give the appropriate users access to the alias. You can do this using the following ESM commands:

```
TSS PER(JOHNS) MQQUEUE(ssid.USE_FOR_GETS) ACC(READ, UPDATE)
TSS PER(JANED) MQQUEUE(ssid.USE_FOR_PUTS) ACC(READ, UPDATE)
```

For BSM, the BSTADMIN commands are:

```
PERMIT MQQUEUE  ssid.USE_FOR_GETS ID(JOHNS) ACC(UPDATE)
PERMIT MQQUEUE  ssid.USE_FOR_PUTS ID(JANED) ACC(UPDATE)
```

This means that user JOHNS is only allowed to get messages from the USE_ALIAS_TO_ACCESS queue through the alias USE_FOR_GETS, and user JANED is only allowed to put messages through the alias queue USE_FOR_PUTS.

If you want to use a technique like this, you must inform the application developers, so that they can design their programs appropriately.

Considerations for model queues

When you open a model queue, WebSphere MQ security makes two queue security checks:

- Are you authorized to access the model queue?

- Are you authorized to access the dynamic queue to which the model queue resolves?

If the dynamic queue name contains a trailing * character, this * is replaced by a character string generated by WebSphere MQ, to create a dynamic queue with a unique name. However, because the whole name, including this generated string, is used for checking authority, you should define generic profiles for these queues.

For example, an MQOPEN call uses a model queue name of CREDIT.CHECK.REPLY.MODEL and a dynamic queue name of CREDIT.REPLY.* on queue manager MQV1. To do this, you must permit the issuing user appropriate access to the following MQQUEUE resources:
MQV1.CREDIT.CHECK.REPLY.MODEL
MQV1.CREDIT.REPLY.*

If you are using BSM, then use the GEN keyword to denote generic profile names. For example:

```
ADD  MQQUEUE  MQV1.CREDIT.CHECK.REPLY.MODEL UACC(NONE)
ADD    MQQUEUE  MQV1.CREDIT.REPLY    GEN      UACC(NONE)
PERMIT MQQUEUE    MQV1.CREDIT.REPLY    GEN  ID(MQU3) ACCESS(UPDATE)
```

Permissions for these resources depend on the type of access required by the MQOPEN call.

A typical dynamic queue name created by an MQOPEN is something like CREDIT.REPLY.20051030163352675. The precise value of the last qualifier is unpredictable; this is why you should use generic profiles for such queue names.

You might also consider defining a profile to control use of the dynamic queue name used by default in the application programming copy members. The WebSphere MQ-supplied copybooks contain a default DynamicQName, which is AMQ.*. This enables an appropriate resource profile to be established.

Close options on permanent dynamic queues

If an application opens a permanent dynamic queue that was created by another application and then attempts to delete that queue with an MQCLOSE option, some extra security checks are applied when the attempt is made. See Table 18.

Table 18. Access levels for close options on permanent dynamic queues

MQCLOSE option	ESM access level required to access ssid.queuename
MQCO_DELETE	ALTER
MQCO_DELETE_PURGE	ALTER

Security and remote queues

When a message is put on a remote queue, a security check is performed against the name of the remote queue. There is no check against the transmission queue identified by the remote queue definition.

This means that users accessing a remote queue need at least UPDATE authority to the resource, because it is not possible to browse a remote queue.

For example, you could define a remote queue as follows (this definition is based on OS/2 formats, and you should use the WebSphere MQ for z/VSE Master Terminal transaction to create appropriate definitions):

```
DEFINE QREMOTE(BANK7.CREDIT.REFERENCE)
       RNAME(CREDIT.SCORING.REQUEST)
       RQMNAME(BNK7)
       XMITQ(BANK1.TO.BANK7)
```

For user JOHNS to put a message to the remote queue, you would need to grant the following access:

```
TSS PER(JOHNS) MQQUEUE(MQV1.BANK7.CREDIT.REFERENCE) ACC(UPDATE)
```

where MQV1 is the local WebSphere MQ for z/VSE subsystem ID.

For BSM, the BSTADMIN command is:

```
PERMIT MQQUEUE  MQV1.BANK7.CREDIT.REFERENCE ID(JOHNS) ACC(UPDATE)
```

Dead-letter queue security

Undelivered messages can be put on a special queue called the dead-letter queue. If you have sensitive data that could possibly be put on this queue, you must consider the security implications of this, because you do not want unauthorized users to be able to retrieve this data.

The only application able to retrieve messages from the dead-letter queue should be a 'special' application that processes the undelivered messages. You can grant access to the dead-letter queue in the same way as any other queue.

The WebSphere MQ for z/VSE Receiver and Sender MCAs user also requires UPDATE authority to the dead-letter queue. The MCAs run as the WebSphere MQ for z/VSE startup user (that is, the user who starts WebSphere MQ for z/VSE using MQSE, MQIT or MQMT). If a message cannot be delivered by either MCA, depending on the channel definition, the MCA may attempt to put the message to the dead-letter queue. Therefore, the MCA user must have UPDATE authority. For example:

```
TSS PER(MQSTART) MQQUEUE(MQV1.DEAD.LETTER.QUEUE) ACC(UPDATE)
```

where MQV1 is the local subsystem ID.

For BSM, the BSTADMIN command is:

```
PERMIT MQQUEUE  MQV1.DEAD.LETTER.QUEUE ID(MQSTART) ACC(UPDATE)
```

If you want to use application programs that can put messages to, or get messages from, the dead-letter queue (or do both), you might consider using aliases, as described in “Using alias queues with MQGET and MQPUT” on page 661.

System queue security

System queues are accessed by the ancillary parts of the queue manager. System queues, in addition to the dead-letter queue, include:

- System log
- System monitor

Messages are put to the system log by the MQER transaction. This transaction runs as either the CICS default user identified by the CICS SIT parameter, or the user specified in the MQER DCT entry (see Chapter 2, “Installation,” on page 13 for more details). Therefore, whichever of these users is configured to put messages to the system log should be granted connection, and also UPDATE authority, to the queue resource.

For example:

```
TSS PER(MQSYS) MQCONN(MQV1.CICS) ACC(READ)
TSS PER(MQSYS) MQQUEUE(MQV1.SYSTEM.LOG) ACC(UPDATE)
```

For BSM, the BSTADMIN commands are:

```
PERMIT MQCONN  MQV1.CICS       ID(MQSYS) ACC(READ)
PERMIT MQQUEUE MQV1.SYSTEM.LOG ID(MQSYS) ACC(UPDATE)
```

For performance reasons, messages written to the system monitor are handled internally to WebSphere MQ for z/VSE. This means that no explicit authority is required for any particular user to put messages to the system monitor queue via normal WebSphere MQ for z/VSE monitoring. If an application needs to explicitly put messages to the system monitor, the application user must have UPDATE authority to the queue resource.

If an application needs to get messages from the system monitor, the application user must have READ or UPDATE authority to the queue resource. For example:

```
TSS PER(JOHNS) MQQUEUE(ssid.SYSTEM.MONITOR) ACC(UPDATE)
```

For BSM, the BSTADMIN command is:

```
PERMIT MQQUEUE ssid.SYSTEM.MONITOR ID(JOHNS) ACC(UPDATE)
```

Reply queue security

WebSphere MQ for z/VSE supports messages with report types of Confirm-On-Arrival (COA) and Confirm-On-Delivery (COD). In either of these cases, a report message is generated by WebSphere MQ for z/VSE. The required user authority varies, depending on whether the report message is for COA or COD, and how the ReplyToQ and ReplyToQMgr fields in the MQMD are used.

User authority for COA: When the ReplyToQ of the object message is a local queue to the z/VSE queue manager, the application user that puts the object message must have UPDATE authority to the ReplyToQ.

When the ReplyToQ of the object message is a remote queue name of the VSE queue manager, the application user that puts the object message must have UPDATE authority to the ReplyToQ.

When the ReplyToQ identifies a local queue on a remote queue manager, and the ReplyToQMgr identifies a remote queue manager, the application user that puts the object message must have UPDATE authority to the remote queue name that resolves the remote local queue and remote queue manager.

User authority for COD: When the ReplyToQ of the object message is a local queue to the z/VSE queue manager, the application user that gets the object message must have UPDATE authority to the ReplyToQ.

When the ReplyToQ of the object message is a remote queue name of the VSE queue manager, the application user that gets the object message must have UPDATE authority to the ReplyToQ.

When the ReplyToQ identifies a local queue on a remote queue manager, and the ReplyToQMgr identifies a remote queue manager, the application user that gets the object message must have UPDATE authority to the remote queue name that resolves the remote local queue and remote queue manager.

User authority for EXPIRY: Message expiry is managed by the queue manager when an application attempts to retrieve a message from a queue. At this time, the

queue manager examines the Expiry field in the message descriptor of the message to determine whether or not the message has expired. If the message has expired, the queue manager continues to search for a valid message to return to the application. Expired messages are never returned.

When a message is identified as 'expired', the queue manager places an expiry entry on transient data queue MQXP. The MQXP data queue is defined at installation time to automatically fire a transaction (also MQXP) when there are items on the queue. For more information about installation and the MQXP transient data queue, refer to “Changing the MQXP TDQ definition” on page 22.

The MQXP transaction is responsible for logically deleting expired messages from queues. It is also responsible for generating expiry report messages when requested.

An expiry report message is requested when the Report field of the message descriptor of the originalmessage indicates one of the following report options:

MQRO_EXPIRATION

MQRO_EXPIRATION_WITH_DATA

MQRO_EXPIRATION_WITH_FULL_DATA

Expiry report messages are sent to the queue identified by the ReplyToQ and ReplyToQMgr message descriptor fields of expired messages. Consequently, the user that runs the MQXP transaction must have connect authority and the authority to put a report message on any potential reply queue.

The MQXP transaction runs as the CICS default user unless the destination control table (DCT) entry for the MQXP transient data queue identifies a specific userid in its definition (see “Changing the MQXP TDQ definition” on page 22 for more details).

Resource definitions for namelist security

If namelist security is active, you must define resources in the MQNLIST class, and permit the necessary groups or user IDs access to these resources, so that they can issue WebSphere MQ API requests that use namelists.

Resource names for namelist security take the form:

ssid.*NamelistName*

where *NamelistName* is the name of the namelist being opened, as specified in the object descriptor on the MQOPEN call.

The ESM/BSM access required to open a namelist is always READ, because namelists can only be opened for inquiry, by way of using the MQOO_INQUIRE option.

Table 19. Access levels for namelist security

MQOPEN option	**ESM access level required to access ssid.NamelistName**
MQOO_INQUIRE	READ

For example, to grant user JOHNS authority to inquire on namelist PAY.QUEUES on queue manager with SSID MQV1:

```
TSS ADD(MQOWNER) MQNLIST(MQV1.PAY.QUEUES)
TSS PER(JOHNS) MQNLIST(MQV1.PAY.QUEUES) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
ADD    MQNLIST MQV1.PAY.QUEUES          UACC(NONE)
PERMIT MQNLIST MQV1.PAY.QUEUES          ID(JOHNS) ACC(READ)
```

Note, that the resource only needs to be created, and ownership applied, once. Therefore, the TSS ADD (or BSM ADD) command is issued only once for each namelist resource defined to class MQNLIST.

Resource definitions for command security

If command security is active, you must define resources in the MQCMDS class, and permit the necessary groups or user IDs access to these resources, so that they can issue WebSphere MQ commands.

Resource names for command security take the form:

```
ssid.command
```

where *command* is a type of command. For example:

```
ssid.ALTER.QLOCAL
ssid.DISPLAY.QMGR
ssid.DELETE.CHANNEL
```

Commands can be issued as:

- PCF messages.
- MQSC verb-based commands.
- Master terminal interactive options.

Command security for PCF messages

The WebSphere MQ command server (long-running transaction MQCS) starts an instance of the WebSphere MQ command processor (transaction MQCX) for each PCF messages retrieved from the system command queue that passes initial validation.

If command security is active, the command server starts with MQCX transaction as the user identified in the UserIdentifier field of the message descriptor of the PCF message. Command security checks are then made for the user running the MQCX transaction. Consequently, command security checks are made against the user running the MQCX transaction, not the user running the MQCS transaction.

For command security to work in this way, the WebSphere MQ command server (MQCS) must be started by a user with surrogate authority for all users that can put messages to the system command queue.

Authority for users that send command messages to the system command queue is three-fold:

- They must have authority to issue the command (for example, DISPLAY).
- They must have authority to send messages the ReplyToQ/ReplyToQMgr.
- They must have authority to issue the command against a specific resource.

This last requirement is only relevant for those commands that manipulate a specific resources, and falls under command resource security described below.

The authority required to issue PCF commands is described in the following table.

Table 20. Command authority for PCF commands

PCF Command	Resource	Authority
MQCMD_CHANGE_CHANNEL	ssid.ALTER.CHANNEL	ALTER
MQCMD_CHANGE_LISTENER	ssid.ALTER.LISTENER	ALTER
MQCMD_CHANGE_Q (alias)	ssid.ALTER.QALIAS	ALTER
MQCMD_CHANGE_SERVICE	ssid.ALTER.SERVICE	ALTER
MQCMD_CHANGE_NAMELIST	ssid.ALTER.NAMELIST	ALTER
MQCMD_CHANGE_Q_MGR	ssid.ALTER.QMGR	ALTER
MQCMD_CHANGE_Q (alias)	ssid.ALTER.QALIAS	ALTER
MQCMD_CHANGE_Q (local)	ssid.ALTER.QLOCAL	ALTER
MQCMD_CHANGE_Q (model)	ssid.ALTER.QMODEL	ALTER
MQCMD_CHANGE_Q (remote)	ssid.ALTER.QREMOTE	ALTER
MQCMD_COPY_CHANNEL	ssid.DISPLAY.CHANNEL ssid.ssid.DEFINE.CHANNEL	READ
MQCMD_COPY_LISTENER	ssid.DISPLAY.LISTENER ssid.DEFINE.LISTENER	READ ALTER
MQCMD_COPY_NAMELIST	ssid.DISPLAY.NAMELIST ssid.DEFINE.NAMELIST	READ ALTER
MQCMD_COPY_Q (alias)	ssid.DISPLAY.QUEUE	READ
	ssid.DEFINE.QALIAS	ALTER
MQCMD_COPY_Q (local)	ssid.DISPLAY.QUEUE	READ
	ssid.DEFINE.QLOCAL	ALTER
MQCMD_COPY_Q (model)	ssid.DISPLAY.QUEUE	READ
	ssid.DEFINE.QMODEL	ALTER
MQCMD_COPY_SERVICE	ssid.DISPLAY.SERVICE ssid.DEFINE.SERVICE	READ ALTER
MQCMD_COPY_Q (remote)	ssid.DISPLAY.QUEUE	READ
	ssid.DEFINE.QREMOTE	ALTER
MQCMD_CREATE_CHANNEL	ssid.DEFINE CHANNEL	ALTER
MQCMD_CREATE_LISTENER	ssid.DEFINE.LISTENER	ALTER
MQCMD_CREATE_NAMELIST	ssid.DEFINE.NAMELIST	ALTER
MQCMD_CREATE_Q (alias)	ssid.DEFINE.QALIAS	ALTER
MQCMD_CREATE_Q (local)	ssid.DEFINE.QLOCAL	ALTER
MQCMD_CREATE_Q (model)	ssid.DEFINE.QMODEL	ALTER
MQCMD_CREATE_Q (remote)	ssid.DEFINE.QALIAS	ALTER
MQCMD_CREATE_SERVICE	ssid.DEFINE.SERVICE	ALTER
MQCMD_DELETE_CHANNEL	ssid.DELETE.CHANNEL	ALTER
MQCMD_DELETE_LISTENER	ssid.DELETE.LISTENER	ALTER
MQCMD_DELETE_NAMELIST	ssid.DELETE.NAMELIST	ALTER
MQCMD_DELETE_Q (alias)	ssid.DELETE.QALIAS	ALTER
MQCMD_DELETE_Q (local)	ssid.DELETE.QLOCAL	ALTER
MQCMD_DELETE_Q (model)	ssid.DELETE.QMODEL	ALTER

Table 20. Command authority for PCF commands (continued)

PCF Command	Resource	Authority
MQCMD_DELETE_Q (remote)	ssid.DELETE.QREMOTE	ALTER
MQCMD_DELETE_SERVICE	ssid.DELETE.SERVICE	ALTER
MQCMD_INQUIRE_CHANNEL	ssid.DISPLAY.CHANNEL	READ
MQCMD_INQUIRE_LISTENER MQCMD_INQUIRE_LISTENER_STATUS	ssid.DISPLAY.LISTENER ssid.DISPLAY.LSSTATUS	READ READ
MQCMD_INQUIRE_CHANNEL_STATUS	ssid.DISPLAY.CHANNEL	READ
MQCMD_INQUIRE_CONNECTION	ssid.DISPLAY.CONN	READ
MQCMD_INQUIRE_Q	ssid.DISPLAY.QUEUE	READ
MQCMD_INQUIRE_Q_MGR	ssid.DISPLAY.QMGR	READ
MQCMD_INQUIRE_Q_STATUS	ssid.DISPLAY.QUEUE	READ
MQCMD_INQUIRE_CHANNEL_NAMES	ssid.DISPLAY.CHANNEL	READ
MQCMD_INQUIRE_NAMELIST	ssid.DISPLAY.NAMELIST	READ
MQCMD_INQUIRE_NAMELIST_NAMES	ssid.DISPLAY.NAMELIST	READ
MQCMD_INQUIRE_Q_NAMES	ssid.DISPLAY.QUEUE	READ
MQCMD_INQUIRE_SERVICE MQCMD_INQUIRE_SERVICE_STATUS	ssid.DISPLAY.SERVICE ssid.DISPLAY.SVSTATUS	READ READ
MQCMD_PING_Q_MGR	ssid.PING.QMGR	CONTROL[1]
MQCMD_RESET_CHANNEL	ssid.RESET.CHANNEL	CONTROL[1]
MQCMD_START_CHANNEL	ssid.START.CHANNEL	CONTROL[1]
MQCMD_START_CHANNEL_LISTENER	ssid.START.LISTENER	CONTROL[1]
MQCMD_START_SERVICE	ssid.START.SERVICE	CONTROL[1]
MQCMD_STOP_CHANNEL	ssid.STOP.CHANNEL	CONTROL[1]
MQCMD_STOP_CONNECTION	ssid.STOP.CONN	CONTROL[1]
MQCMD_STOP_LISTENER MQCMD_STOP_SERVICE	ssid.STOP.LISTENER ssid.STOP.SERVICE	CONTROL[1] CONTROL[1]

Note:

1. If you are using the Basic Security Manager (BSM), then use authority UPDATE instead of CONTROL.

Command security for WebSphere MQ commands

WebSphere MQ commands are generated by the WebSphere MQ MQSC command utility. The MQSC utility generates PCF Escape messages from SYSIPT batch input and places them on the system command queue using the WebSphere MQ batch interface. Consequently, the user that runs the MQSC command utility must have authority to connect to the queue manager and put messages on the system command queue.

Once an MQSC command has been placed on the system command queue (as a PCF Escape message) it is treated like any other PCF message. The WebSphere MQ command server retrieves the message and starts the WebSphere MQ command processor (MQCX transaction) as the user identified in the UserIdentifier field of the message descriptor.

If security is active, the user that submits the batch job to run the MQSC command utility is the userid that is placed in the UserIdentifier field. This user must have

authority to put a reply message on the system reply queue. The system reply queue name is configurable as a communication parameter of the global system definition.

The authority required for PCF Escape messages is dependent on the verb-based text of the MQSC command embedded in the Escape message, and is determined by the following table:

Table 21. Command authority for WebSphere MQ commands

WebSphere MQ Command	Resource	Authority
ALTER CHANNEL	ssid.ALTER.CHANNEL	ALTER
ALTER LISTENER	ssid.ALTER.LISTENER	ALTER
ALTER NAMELIST	ssid.ALTER.NAMELIST	ALTER
ALTER QMGR	ssid.ALTER.QMGR	ALTER
ALTER QALIAS	ssid.ALTER.QALIAS	ALTER
ALTER QLOCAL	ssid.ALTER.QLOCAL	ALTER
ALTER QMODEL	ssid.ALTER.QMODEL	ALTER
ALTER QREMOTE	ssid.ALTER.QREMOTE	ALTER
ALTER SERVICE	ssid.ALTER.SERVICE	ALTER
DEFINE CHANNEL	ssid.DEFINE CHANNEL	ALTER
DEFINE LISTENER	ssid.DEFINE.LISTENER	ALTER
DEFINE NAMELIST	ssid.DEFINE.NAMELIST	ALTER
DEFINE QALIAS	ssid.DEFINE.QALIAS	ALTER
DEFINE QLOCAL	ssid.DEFINE.QLOCAL	ALTER
DEFINE QMODEL	ssid.DEFINE.QMODEL	ALTER
DEFINE QREMOTE	ssid.DEFINE.QALIAS	ALTER
DEFINE SERVICE	ssid.DEFINE.SERVICE	ALTER
DELETE CHANNEL	ssid.DELETE.CHANNEL	ALTER
DELETE LISTENER	ssid.DELETE.LISTENER	ALTER
DELETE NAMELIST	ssid.DELETE.NAMELIST	ALTER
DELETE QALIAS	ssid.DELETE.QALIAS	ALTER
DELETE QLOCAL	ssid.DELETE.QLOCAL	ALTER
DELETE QMODEL	ssid.DELETE.QMODEL	ALTER
DELETE QREMOTE	ssid.DELETE.QREMOTE	ALTER
DELETE SERVICE	ssid.DELETE.SERVICE	ALTER
DISPLAY CHANNEL	ssid.DISPLAY.CHANNEL	READ
DISPLAY CHSTATUS	ssid.DISPLAY.CHANNEL	READ
DISPLAY CONN	ssid.DISPLAY.CONN	READ
DISPLAY LISTENER DISPLAY LSSTATUS	ssid.DISPLAY.LISTENER ssid.DISPLAY.LSSTAUS	READ READ
DISPLAY NAMELIST	ssid.DISPLAY.NAMELIST	READ
DISPLAY QALIAS	ssid.DISPLAY.QUEUE	READ
DISPLAY QLOCAL	ssid.DISPLAY.QUEUE	READ
DISPLAY QMODEL	ssid.DISPLAY.QUEUE	READ

Table 21. Command authority for WebSphere MQ commands (continued)

WebSphere MQ Command	Resource	Authority
DISPLAY QREMOTE	ssid.DISPLAY.QUEUE	READ
DISPLAY QMGR	ssid.DISPLAY.QMGR	READ
DISPLAY QSTATUS	ssid.DISPLAY.QUEUE	READ
DISPLAY SERVICE	ssid.DISPLAY.SERVICE	READ
DISPLAY SVSTATUS	ssid.DISPLAY.SVSTATUS	READ
PING QMGR	ssid.PING.QMGR	CONTROL[1]
RESET CHANNEL	ssid.RESET.CHANNEL	CONTROL[1]
START CHANNEL	ssid.START.CHANNEL	CONTROL[1]
START LISTENER	ssid.START.LISTENER	CONTROL[1]
START SERVICE	ssid.START.SERVICE	CONTROL1
STOP CHANNEL	ssid.STOP.CHANNEL	CONTROL[1]
STOP CONN	ssid.STOP.CONN	CONTROL[1]
STOP LISTENER	ssid.STOP.LISTENER	CONTROL1
STOP SERVICE	ssid.STOP.SERVICE	CONTROL1

Note:

1. If you are using the Basic Security Manager (BSM), then use authority ALTER instead of CONTROL.

Command security for MQMT options

Command can be issued interactively via the WebSphere MQ master terminal transaction. Generally, these are invoked in native CICS from the MQMT transaction.

The MQMT transaction provides a primary options menu. Most of the menu options can be invoked directly by starting the appropriate WebSphere MQ transaction.

The following table describes these options and transactions, and the resources and authority necessary to perform the option.

Table 22. Command authority for MQMT options

Option	Trans	Function	Resource	Authority
1.1	MQMS	Maintain QMgr	ssid.DISPLAY.QMGR	READ
			ssid.ALTER.QMGR	ALTER
1.2	MQMQ	Maintain Queues	ssid.DISPLAY.QUEUE	READ
1.2	MQMQ	Maintain Queue (alias)	ssid.ALTER.QALIAS	ALTER
1.2	MQMQ	Maintain Queue (local)	ssid.ALTER.QLOCAL	ALTER
1.2	MQMQ	Maintain Queue (remote)	ssid.ALTER.QREMOTE	ALTER
1.3	MQMH	Maintain Channel	ssid.DISPLAY.CHANNEL	READ
			ssid.ALTER.CHANNEL	ALTER
1.5	MQMN	Maintain Namelists	ssid.DISPLAY.NAMELIST ssid.ALTER.NAMELIST	READ ALTER
1.6	MQDS	Display QMgr	ssid.DISPLAY.QMGR	READ

Table 22. Command authority for MQMT options (continued)

Option	Trans	Function	Resource	Authority
1.7	MQDQ	Display Queues	ssid.DISPLAY.QUEUE	READ
1.8	MQDH	Display Channel	ssid.DISPLAY.CHANNEL	READ
1.10	MQDN	Display Namelists	ssid.DISPLAY.NAMELIST	READ
2.2	MQMB	Start channel	ssid.START.CHANNEL	CONTROL[1]
2.2	MQMB	Stop channel	ssid.STOP.CHANNEL	CONTROL[1]
2.3	MQMR	Reset channel	ssid.RESET.CHANNEL	CONTROL[1]
3.1	MQQM	Monitor queues	ssid.DISPLAY.QUEUE	READ
3.2	MQCM	Monitor channel	ssid.DISPLAY.CHANNEL	READ

Note:

1. If you are using the Basic Security Manager (BSM), then use authority ALTER instead of CONTROL.

In addition to the command authority required to issue a command, the issuing user must have command resource authority for a specific resource when a specific resource is affected by the command.

Table 23 describes the resources and authority necessary to perform the various actions on listener and service objects using the administrator panels.

Table 23. Authority and profiles for listener and service objects

Panel actions	MQCMDS profile	Access	MQADMIN profile
Update	ssid.ALTER.LISTENER	ALTER	ssid.LISTENER.listener
Update	ssid.ALTER.SERVICE	ALTER	ssid.SERVICE.service
Add	ssid.DEFINE.LISTENER	ALTER	ssid.LISTENER.listener
Add	ssid.DEFINE.SERVICE	ALTER	ssid.SERVICE.service
Delete	ssid.DELETE.LISTENER	ALTER	ssid.LISTENER.listener
Delete	ssid.DELETE.SERVICE	ALTER	ssid.SERVICE.service
Read	ssid.DISPLAY.LISTENER	READ	No check
Read	ssid.DISPLAY.SERVICE	READ	No check
Start	ssid.START.LISTENER	CONTROL[1]	No check
Start	ssid.START.SERVICE	CONTROL[1]	No check
Stop	ssid.STOP.LISTENER	CONTROL[1]	No check
Stop	ssid.STOP.SERVICE	CONTROL[1]	No check

Note:

1. If you are using the Basic Security Manager (BSM), then use authority ALTER instead of CONTROL.

Resource definitions for command resource security

If command resource security is active, you must grant command authority by resource to any user that is authorized to issue commands against that specific resource.

Command security allows a user to issue certain commands. Command resource security allows a user to issue those commands against specific resources. For

example, a user may be authorized to 'DISPLAY' certain queues and not others. To achieve this, the user is granted command authority to 'DISPLAY', and is then granted command resource security for each queue the user is allowed to display.

Resources relevant to command resource security must be defined in the MQADMIN class.

The following table describes the resources and authority required for PCF messages (the constant 'ssid' should be replaced by the queue manager's subsystem ID, and names expressed in lower-case should be replaced with the names of specific resources):

Table 24. Command resource authority for PCF commands

Command	Command resource	Authority
MQCMD_CHANGE_CHANNEL	ssid.CHANNEL.channel	ALTER
MQCMD_CHANGE_NAMELIST	ssid.NAMELIST.namelist	ALTER
MQCMD_CHANGE_Q	ssid.QUEUE.queue	ALTER
MQCMD_COPY_CHANNEL	ssid.CHANNEL.tochannel	ALTER
MQCMD_COPY_NAMELIST	ssid.NAMELIST.namelist	ALTER
MQCMD_COPY_Q	ssid.QUEUE.toqueue	ALTER
MQCMD_CREATE_CHANNEL	ssid.CHANNEL.channel	ALTER
MQCMD_CREATE_NAMELIST	ssid.NAMELIST.namelist	ALTER
MQCMD_CREATE_Q	ssid.QUEUE.queue	ALTER
MQCMD_DELETE_CHANNEL	ssid.CHANNEL.channel	ALTER
MQCMD_DELETE_NAMELIST	ssid.NAMELIST.namelist	ALTER
MQCMD_DELETE_Q	ssid.QUEUE.queue	ALTER
MQCMD_RESET_CHANNEL	ssid.CHANNEL.channel	CONTROL[1]
MQCMD_START_CHANNEL	ssid.CHANNEL.channel	CONTROL[1]
MQCMD_STOP_CHANNEL	ssid.CHANNEL.channel	CONTROL[1]

Note:

1. if you are using the Basic Security Manager (BSM), then use authority ALTER instead of CONTROL.

The following table describes the resources and authority required for MQCS commands (the constant 'ssid' should be replaced by the queue manager's subsystem identifier, and names expressed in lower-case should be replaced with the names of specific resources):

Table 25. Command resource authority for WebSphere MQ commands

MQCS Command	Resource	Authority
ALTER CHANNEL	ssid.CHANNEL.channel	ALTER
ALTER NAMELIST	ssid.NAMELIST.namelist	ALTER
ALTER QALIAS	ssid.QUEUE.queue	ALTER
ALTER QLOCAL	ssid.QUEUE.queue	ALTER
ALTER QMODEL	ssid.QUEUE.queue	ALTER
ALTER QREMOTE	ssid.QUEUE.queue	ALTER
DEFINE CHANNEL	ssid.CHANNEL.channel	ALTER

Table 25. Command resource authority for WebSphere MQ commands (continued)

MQCS Command	Resource	Authority
DEFINE NAMELIST	ssid.NAMELIST.namelist	ALTER
DEFINE QALIAS	ssid.QUEUE.queue	ALTER
DEFINE QLOCAL	ssid.QUEUE.queue	ALTER
DEFINE QMODEL	ssid.QUEUE.queue	ALTER
DEFINE QREMOTE	ssid.QUEUE.queue	ALTER
DELETE CHANNEL	ssid.CHANNEL.channel	ALTER
DELETE NAMELIST	ssid.NAMELIST.namelist	ALTER
DELETE QALIAS	ssid.QUEUE.queue	ALTER
DELETE QLOCAL	ssid.QUEUE.queue	ALTER
DELETE QMODEL	ssid.QUEUE.queue	ALTER
DELETE QREMOTE	ssid.QUEUE.queue	ALTER
RESET CHANNEL	ssid.CHANNEL.channel	CONTROL[1]
START CHANNEL	ssid.CHANNEL.channel	CONTROL[1]
STOP CHANNEL	ssid.CHANNEL.channel	CONTROL[1]

Note:

1. if you are using the Basic Security Manager (BSM), then use authority ALTER instead of CONTROL.

The following table describes the resources and authority required for master terminal options and associated transactions (the constant 'ssid' should be replaced by the queue manager's subsystem identifier, and names expressed in lower-case should be replaced with the names of specific resources):

Table 26. Command resource authority for MQMT options

Option	Trans	Function	Resource	Authority
1.2	MQMQ	Maintain Queues	ssid.QUEUE.queue	ALTER
1.3	MQMH	Maintain Channel	ssid.CHANNEL.channel	ALTER
1.5	MQMN	Maintain Namelists	ssid.NAMELIST.namelist	ALTER
2.2	MQMB	Start channel	ssid.CHANNEL.channel	CONTROL[1]
2.2	MQMB	Stop channel	ssid.CHANNEL.channel	CONTROL[1]
2.3	MQMR	Reset channel	ssid.CHANNEL.channel	CONTROL[1]

Note:

1. If you are using the Basic Security Manager (BSM), then use authority UPDATE instead of CONTROL.

In addition to the command resource authority described in table x, the master terminal transactions also require special authority for options 2.5 and 4.0. Relevant resources must be defined in the MQQUEUE class (not the MQADMIN class) according to the following table:

Table 27. Command resource authority for MQMT options 2.5 and 4.0

Option	Trans	Function	Resource	Authority
2.5	MQMD	Maintain messages	ssid.queue	UPDATE

Table 27. Command resource authority for MQMT options 2.5 and 4.0 (continued)

Option	Trans	Function	Resource	Authority
4.0	MQBQ	Browse queues	ssid.queue	READ

Security implementation checklist

This section contains a step-by-step procedure you can use to work out and define the security implementation for each of your WebSphere MQ subsystems. Refer to other sections for details, in particular "Using security classes and resources" on page 655.

If you require security checking to be implemented on at least one of your WebSphere MQ subsystems, you must first activate the MQADMIN class. Then, for each WebSphere MQ subsystem, you must decide whether you need security checking on that subsystem. If you do not require security checking, you must define an ssid.NO.SUBSYS.SECURITY profile in the MQADMIN class.

If you do require security checking, use the following checklist to implement it:

- Do you need connection security?

  Yes: Define appropriate connection profiles in the MQCONN class and permit the appropriate users or groups access to these profiles.

  No: Define an ssid.NO.CONNECT.CHECKS resource in the MQADMIN class and grant your WebSphere MQ for z/VSE startup user READ access to the resource.

- Do you need security checking on commands?

  Yes: Activate the MQCMDS class. Define appropriate command profiles in the MQCMDS class and permit the appropriate users access to these profiles. If command authority by individual resource is required, define appropriate resource profiles in the MQADMIN class and grant access to these profiles as appropriate.

  No: Define resources ssid.NO.CMDS.CHECKS and ssid.NO.CMD.RESC.CHECKS in the MQADMIN class and grant read authority to both resources to your WebSphere MQ for z/VSE startup user.

- Do you need queue security?

  Yes: Activate the MQQUEUE class. Define appropriate queue resources in the MQQUEUE class and permit the appropriate user access to these profiles.

  Also, ensure that your WebSphere MQ VSAM datasets are protected against unauthorized access.

  No: Define an ssid.NO.QUEUE.CHECKS profile in the MQADMIN class and grant read authority to your WebSphere MQ for z/VSE startup user.

- Do you need namelist security?

  Yes: Activate the MQNLIST class. Define appropriate namelist resources in the MQNLIST class and permit the appropriate user access to these profiles.

  No: Define an ssid.NO.NLIST.CHECKS profile in the MQADMIN class and grant read authority to your WebSphere MQ for z/VSE startup user.

- Do you plan to have remote client connections?

Yes: Ensure that your remote clients use the MQ_USER_ID and MQ_PASSWORD environment variables, and ensure that such users are defined to your ESM or BSM.

For a detailed example of resource definitions and access authorities for WebSphere MQ for z/VSE, refer to Appendix H, “Security implementation,” on page 1055.

Chapter 13. API exits

API exits let you write code that changes the behavior of WebSphere MQ API calls, such as MQPUT and MQGET, and then insert that code immediately before or immediately after those calls. The insertion is automatic; the queue manager drives the exit code at the registered points.

This chapter explains why you might want to use API exits, then describes what administration tasks are involved in enabling them. The sections are:

Why you would use API exits

There are many reasons why you might want to insert code that modifies the behavior of applications at the level of the queue manager. Each of your applications has a specific job to do, and its code should do that task as efficiently as possible. At a higher level, you might want to apply standards or business processes to a particular queue manager for all the applications that use that queue manager. It is more efficient to do this above the level of individual applications, and thus without having to change the code of each application affected.

Here are a few suggestions of areas in which API exits might be useful:

- For security, you can provide authentication, checking that applications are authorized to access a queue or queue manager. You can also police applications' use of the API, authenticating the individual API calls, or even the parameters they use.
- For flexibility, you can respond to rapid changes in your business environment without changing the applications that rely on the data in that environment. You could, for example, have API exits that respond to changes in interest rates, currency exchange rates, or the price of components in a manufacturing environment.
- For monitoring use of a queue or queue manager, you can trace the flow of applications and messages, log errors in the API calls, set up audit trails for accounting purposes, or collect usage statistics for planning purposes.

Configuring API exits

On WebSphere MQ for z/VSE, API exits are configured as part of the queue manager's global system definition. API exits can only be configured using the WMQ master terminal transactions; there is no facility to configure API exits using PCF, MQSC or the WebSphere MQ Explorer on z/VSE.

Using the master terminal transaction MQMT option 1.1, and then PF12, you can start the API Exits configuration screen.

```
10/22/2008        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
10:51:25                   Global System Definition                 CIC1
MQWMSYS                       API Exits Settings                    A000

   Local API Exits
   1. Name: MQ_SAMPLE_EXIT
      Module: MQPSAXE     Data: Exit data goes here if needed.
   2. Name:
      Module:             Data:
   3. Name:
      Module:             Data:
   4. Name:
      Module:             Data:
   5. Name:
      Module:             Data:
   6. Name:
      Module:             Data:
   7. Name:
      Module:             Data:
   8. Name:
      Module:             Data:
Requested record displayed.
PF2=Queue Manager details  PF3=Quit   PF4/Enter=Read   PF6=Update
```

Figure 86. API Exits screen

On this screen, you can configure up to eight API exits and their associated exit data. If more than one exit is configured, the API exits are called in the order enumerated on the screen.

Here are the entry fields on this screen:

Name The descriptive name of the API exit passed to it in the ExitInfoName field of the MQAXP structure. Like all WMQ object names, it must conform to the naming rules described in “Object names” on page 3.

The maximum length of this field is MQ_OBJECT_NAME_LENGTH.

Module

The executable phase that contains your MQ API exit routine. API exits programs must be accessible to your CICS region.

The maximum length of this field is MQ_EXIT_NAME_LENGTH,

Data Exit data passed to your exit program on invocation.

The maximum length of this field is MQ_EXIT_DATA_LENGTH.

Use the PF6 function key to add or update your MQ API exit settings.

How API exits work

The API exits defined to your queue manager are called in the sequence specified when an MQCONN/MQCONNX call is issued by the queue manager or an application. The API exit is expected to use the MQXEP application program interface to register other exit routines that are to be called either before or after MQI calls.

The MQXEP API is provided in the WMQ for z/VSE installation library as file MQXEP.OBJ, and is described in detail in “MQXEP - Register entry point” on page 696. This object must be link-edited with your API exit program.

Your API exit can use MQXEP to register routines to be called before and after the following:

- MQCONN/MQCONNX, to provide a queue manager connection handle for use on subsequent API calls.
- MQDISC, to disconnect from a queue manager.
- MQBACK, to back out a UOW.
- MQCMIT, to commit a UOW.
- MQOPEN, to open an WMQ resource for subsequent access.
- MQCLOSE, to close an WMQ resource that had previously been opened for access.
- MQGET, to retrieve a message from a queue that has previously been opened for access.
- MQPUT, to place a message on to a queue that has previously been opened for access.
- MQINQ, to inquire on the attributes of an WMQ resource that has previously been opened for access.
- MQSET, to set the attributes of a queue that has previously been opened for access.

Note that WebSphere MQ for z/VSE does not support exit points for the MQPUT1 call. This is because on WMQ for z/VSE the MQPUT1 call is an encapsulation of the MQOPEN, MQPUT and MQCLOSE calls. Consequently, if you need to intercept the MQPUT1 call, you must at least intercept the MQPUT call.

In addition, MQXEP can be called to register an exit routine before message data conversion performed during MQGET processing that uses the MQGMO_CONVERT option.

Registering exit points involves identifying the relevant MQI call, whether the exit function should be called before or after the MQI call, and the address of the exit function to call. These are passed as parameters to the MQXEP call.

Once the API exit (or exits) has been called by the queue manager, it is not called again until another MQCONN/MQCONNX call is issued. Only the exit functions registered by the API exit are called thereafter. These are called as often as the relevant MQI calls are issued for the connection that activated the API exit. When the connection ends, via the MQDISC call, the exit functions registered by the API exit are deregistered automatically.

Exit functions registered with MQXEP can also call the MQXEP API to deregister themselves or other exit functions associated with the connection. For more information about deregistering exit functions refer to “MQXEP - Register entry point” on page 696.

How to write an API exit

You write your exits using the C, COBOL or PL/I programming languages. To help you do so, we provide a sample exit, MQPSAXE.Z, that generates trace entries to a named file. When you start writing exits, we recommend that you use this as your starting point.

Essentially, your API exit program will be a series of calls to the MQXEP API that register the MQI calls you want to intercept and whether the interception should occur before and/or after the MQI calls.

For example, the following C language code illustrates how you might register an exit function to be called before each MQOPEN call.

```
#include <cmqc.h>
#include <cmqxc.h>

#pragma linkage(EntryPoint, FETCHABLE)
MQ_INIT_EXIT EntryPoint;
void MQENTRY EntryPoint ( PMQAXP  pExitParms
      , PMQAXC  pExitContext
      , PMQLONG pCompCode
      , PMQLONG pReason )
{
    PMQFUNC  xfunc;

    xfunc = (PMQFUNC)fetchep((void(*)())OpenBefore);

    MQXEP ( pExitParms->Hconfig
    , MQXR_BEFORE
    , MQXF_OPEN
    , xfunc
    , NULL
    , pCompCode
    , pReason );

    return;
}

MQ_OPEN_EXIT OpenBefore;
void MQENTRY OpenBefore ( PMQAXP   pExitParms
      , PMQAXC   pExitContext
      , PMQHCONN pHconn
      , PPMQOD   ppObjDesc
      , PMQLONG  pOptions
      , PPMQHOBJ ppHobj
      , PMQLONG  pCompCode
      , PMQLONG  pReason )
{
    .
    .
    .
    return;
}
```

In this example, the API exit entry point called EntryPoint is called when the queue manager or an application issues an MQCONN/MQCONNX call. The API exit calls MQXEP to register exit function OpenBefore to be called before each MQOPEN call. Consequently, each call to MQOPEN associated with the connection that activated the API exit will result in a call to the OpenBefore function. Calls to OpenBefore end when the MQDISC is called for the connection.

Note that your API exit and registered exit functions are always passed the MQAXP and MQAXC data structures. These are described in “MQAXP - API exit parameter” on page 689 and “MQAXC - API exit context” on page 685.

The following table identifies the copybooks and header files containing the constants and prototypes needed to write API exits. These are provided in the WMQ for z/VSE installation sublibrary.

Table 28. API exit copybooks

Copybook	Description
CMQXC.H	C language header file

Table 28. API exit copybooks (continued)

Copybook	Description
CMQXP.P	PL/I language include file
CMQXV.C	COBOL copybook

Exit functions take the general form:

```
MQ_call_EXIT (parameters)
```

where *call* is the API call name (PUT, GET, and so on), and the parameters control the function of the exit, primarily providing communication between the exit and the external control blocks MQAXP (the exit parameter structure) and MQAXC (the exit context structure).

Compiling API exits

API exits can be written in C, COBOL or PL/I programming languages. The MQ_INIT_EXIT entry point must be fetchable. For example, in the C language you can use the following compiler directive:

```
#pragma linkage(my_init_exit, FETCHABLE)
```

When an MQCONN/MQCONNX call is issued, the queue manager uses the C run-time FETCH() call to load configured API exits into storage, and then branches to the fetched entry point of your API exit. The API exit is called prior to the "before" MQCONN/MQCONNX exit point. Consequently, if your API exit uses MQXEP to register an exit function to be called before MQCONN/MQCONNX, it will be called after the API exit has run.

You compile your API exit with your relevant Language Environment for z/VSE compiler.

Linking API exits

API exits must be linked with the MQXEP.OBJ object file provided in the WMQ for z/VSE installation sublibrary.

The entry point in the MQXEP.OBJ file is also called MQXEP, consequently you do not need an INCLUDE card in you link or pre-link step.

API Exit reference information

This section provides reference information for the API exit. It includes:

The data structures, calls, and exits are described in the order shown above (alphabetic order within each type).

General usage notes

This section contains general usage notes that relate to all API exit functions.

1. All exit functions can issue the MQXEP call; this call is designed specifically for use from API exit functions.
2. The MQ_INIT_EXIT function cannot issue any WMQ calls other than MQXEP.
3. All other exit functions can issue the following MQ calls: MQBACK, MQCLOSE, MQCMIT, MQCONN, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET.
4. If an exit function issues the MQCONN call, the call completes with reason code MQRC_ALREADY_CONNECTED, and the handle returned is the same as the one passed to the exit as a parameter.
5. When an API exit function issues an MQI call, API exits are not called recursively. Consequently, if an exit issues an MQI call, no exit will be called for that MQI call.
6. Exit functions can also put and get messages within the application's unit of work. When the application commits or backs out the unit of work, all messages within the unit of work are committed or backed out together, regardless of who placed them in the unit of work (application or exit function).

   When an exit function uses the application's unit of work in this way, the exit function should usually avoid issuing the MQCMIT call, as this commits the application's unit of work and may impair the correct functioning of the application. However, the exit function may sometimes need to issue the MQBACK call, if the exit function encounters a serious error that prevents the unit of work being committed (for example, an error putting a message as part of the application's unit of work). When MQBACK is called, take care to ensure that the application unit of work boundaries are not changed. In this situation the exit function must set the appropriate values to ensure that completion code MQCC_FAILED and reason code MQRC_BACKED_OUT are returned to the application, so that the application can detect the fact that the unit of work has been backed out.

   If an exit function uses the application's connection handle to issue MQ calls, those calls do not themselves result in further invocations of API exit functions.
7. If an MQXR_BEFORE exit function terminates abnormally, the queue manager may be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated.
8. If an MQXR_AFTER exit function terminates abnormally, the queue manager may be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated. Be aware that in

the latter case, messages retrieved outside a unit of work are lost (this is the same situation as the application failing immediately after removing a message from the queue).

MQACH - API exit chain header

This table summarizes the fields in the structure.

Table 29. Fields in MQACH

Field	Description
StrucId	Structure identifier
Version	Structure version number
StrucLength	Length of MQACH structure
ChainAreaLength	Total length of chain area
ExitInfoName	Exit information name
NextChainAreaPtr	Address of next chain area

The MQACH structure describes the header information that must be present at the start of each exit chain area.

- The address of the first area in the chain in given by the ExitChainAreaPtr field in MQAXP. If there is no chain, ExitChainAreaPtr is the null pointer.
- The address of the next area in the chain is given by the NextChainAreaPtr field in MQACH. For the last area in the chain, NextChainAreaPtr is the null pointer.

Any exit function can create a chain area in dynamically-obtained storage (for example, by using GETMAIN), and add that area to the chain at the desired location (start, middle, or end). The exit function must ensure that it sets all fields in MQACH to valid values.

The exit suite that creates the chain area is responsible for destroying that chain area before termination (the MQ_TERM_EXIT function is a convenient point at which to do this). However, adding and removing chain areas from the chain must be done only by an exit function when it is invoked by the queue manager; this restriction is necessary to avoid serialization problems.

Exit chain areas are made available to all exit suites, and must not be used to hold private data. Use ExitUserArea in MQAXP to hold private data.

In general there is no correspondence between the chain of exit functions that are invoked for an API call, and the chain of exit chain areas:

- Some exit functions might not have chain areas.
- Other exit functions might each have multiple chain areas.
- The order of the chain areas might be different from the order of the exit functions that own those chain areas.

Fields

The MQACH structure contains these fields:

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQACH_STRUC_ID
Identifier for API exit chain header structure.

For the C programming language, the constant MQACH_STRUC_ID_ARRAY is also defined; this has the same value as MQACH_STRUC_ID, but is an array of characters instead of a string.

This initial value of this field is MQACH_STRUC_ID.

Version (MQLONG)
Structure version number.

The value is:

MQACH_VERSION_1
Version-1 API exit chain header structure.

The following constant specifies the version number of the current version:

MQACH_CURRENT_VERSION
Current version of API exit chain header structure.

Note: When a new version of the MQACH structure is introduced, the layout of the existing part is not changed. The exit function must therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit function needs to use.

The initial value of this field is MQACH_CURRENT_VERSION.

StrucLength (MQLONG)
Length of MQACH structure.

This is the length of the MQACH structure itself; this length excludes the exit-defined data that follows the MQACH structure (see the ChainAreaLength field).

- The exit function that creates the MQACH structure must set this field to the length of the MQACH.
- An exit function that wants to access the exit-defined data should use StrucLength as the offset of the exit-defined data from the start of the MQACH structure.

The following value is defined:

MQACH_LENGTH_1
Length of version-1 MQACH structure.

The following constant specifies the length of the current version:

MQACH_CURRENT_LENGTH
Length of current version of exit chain area header.

The initial value of this field is MQACH_CURRENT_LENGTH.

ChainAreaLength (MQLONG)
Total length of chain area.

This is the total length of the chain area. It is equal to the sum of the length of the MQACH plus the length of the exit-defined data that follows the MQACH.

The initial value of this field is zero.

ExitInfoName (MQCHAR48)
Exit information name.

This is a name that is used to identify the exit suite to which the chain area belongs.

The length of this field is given by MQ_EXIT_INFO_NAME_LENGTH.

The initial value of this field is the null string in C.

NextChainAreaPtr (PMQACH)
Address of next MQACH structure in chain.

This is the address of the next chain area in the chain. If the current chain area is the last one in the chain, NextChainAreaPtr is the null pointer.

The initial value of this field is the null pointer.

C declaration

```
typedef struct tagMQACH MQACH;
struct tagMQACH
{
  MQCHAR4   StrucId;    /* Structure identifier */
  MQLONG    Version;    /* Structure version number */
  MQLONG    StrucLength;    /* Length of MQACH structure */
  MQLONG    ChainAreaLength;  /* Total length of chain area */
  MQCHAR48  ExitInfoName;     /* Exit information name */
  PMQACH    NextChainAreaPtr;   /* Address of next MQACH structure in chain */
};
```

MQAXC - API exit context

The following table summarizes the fields in the structure.

Table 30. Fields in MQAXC

Field	Description
StrucId	Structure identifier
Version	Structure version number
Environment	Environment
UserId	User identifier
SecurityId	Security identifier
ConnectionName	Connection name
LongMCAUserIdLength	Length of long MCA user identifier
LongRemoteUserIdLength	Length of long remote user identifier
LongMCAUserIdPtr	Address of long MCA user identifier
LongRemoteUserIdPtr	Address of long remote user identifier
ApplName	Application name
ApplType	Application type
ProcessId	Process identifier
ThreadId	Thread identifier

The MQAXC structure describes the context information that is passed to an API exit. The context information relates to the environment in which the application is running.

Fields

The MQAXC structure contains these fields:

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQAXC_STRUC_ID
Identifier for API exit parameter structure.

For the C programming language, the constant MQAXC_STRUC_ID_ARRAY is also defined; this has the same value as MQAXC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

The value is:

MQAXC_VERSION_1
Version-1 API exit parameter structure.

The following constant specifies the version number of the current version:

MQAXC_CURRENT_VERSION
Current version of API exit parameter structure.

Note: When a new version of the MQAXC structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

Environment (MQLONG)
Environment.

This indicates the environment from which the API call was issued. The value is one of the following:

MQXE_COMMAND_SERVER
Command server.

MQXE_MCA
Message channel agent.

MQXE_OTHER
Environment not defined.

This is an input field to the exit.

UserId (MQCHAR12)
User identifier.

This is the user identifier associated with the program that issued the API call. For a client connection (MQXE_MCA_SVRCONN), UserId contains the user identifier of the adopted user, and not the user identifier of the MCA.

The length of this field is given by MQ_USER_ID_LENGTH.

This is an input field to the exit.

SecurityId (MQBYTE40)
Security identifier.

This is the security identifier associated with the program that issued the API call. For a client connection (MQXE_MCA_SVRCONN), SecurityId contains the security identifier of the adopted user, and not the security identifier of the MCA. If the security identifier is not known, SecurityId has the value:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this has the same value as MQSID_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_SECURITY_ID_LENGTH. This is an input field to the exit.

ConnectionName (MQCHAR264)
Connection name.

For a client connection (MQXE_MCA_SVRCONN), this field contains the address of the client (for example, the TCP/IP address). In other cases, this field is blank.

The length of this field is given by MQ_CONN_NAME_LENGTH.

This is an input field to the exit.

LongMCAUserIdLength (MQLONG)
Length of long MCA user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the length in bytes of the full MCA user identifier pointed to by LongMCAUserIdPtr. In other cases, this field is zero.

This is an input field to the exit.

LongRemoteUserIdLength (MQLONG)
Length of long remote user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the length in bytes of the full remote user identifier pointed to by LongRemoteUserIdPtr. In other cases, this field is zero.

This is an input field to the exit.

LongMCAUserIdPtr (MQPTR)
Address of long MCA user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the address of the full MCA user identifier. The length of the full identifier is given by LongMCAUserIdLength. In other cases, this field is the null pointer.

This is an input field to the exit.

LongRemoteUserIdPtr (MQPTR)
Address of long remote user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the address of the full remote user identifier. The length of the full identifier is given by LongRemoteUserIdLength. In other cases, this field is the null pointer.

This is an input field to the exit.

ApplName (MQCHAR28)
Application name.

This is the name of the application that issued the API call. For WebSphere MQ for z/VSE, this is the CICS transaction id. The length of this field is given by MQ_APPL_NAME_LENGTH.

This is an input field to the exit.

ApplType (MQLONG)
Application type.

This is the type of the application that issued the API call. The value is the same as MQAT_DEFAULT for the environment for which the application was compiled.

This is an input field to the exit.

ProcessId (MQPID)
The WebSphere MQ process identifier.

For WebSphere MQ for z/VSE, this is the CICS task number.

This is an input field to the exit.

ThreadId (MQTID)
The WebSphere MQ thread identifier.

For WebSphere MQ for z/VSE, this is the API exit sequence number. For example, if the exit being passed this parameter is the fifth exit in the API exit chain, then the ThreadId is 5.

This is an input field to the exit.

C declaration

```
typedef struct tagMQAXC MQAXC;
struct tagMQAXC
{
  MQCHAR4   StrucId;      /* Structure identifier */
  MQLONG    Version;      /* Structure version number */
  MQLONG    Environment;      /* Environment */
  MQCHAR12  UserId;       /* User identifier */
  MQBYTE40  SecurityId;       /* Security identifier */
  MQCHAR264 ConnectionName;     /* Connection name */
  MQLONG    LongMCAUserIdLength;    /* Length of long MCA user identifier */
  MQLONG    LongRemoteUserIdLength; /* Length of long remote user identifier */
  MQPTR     LongMCAUserIdPtr;     /* Address of long MCA user identifier */
  MQPTR     LongRemoteUserIdPtr;    /* Address of long remote user identifier */
  MQCHAR28  ApplName;       /* Application name */
  MQLONG    ApplType;       /* Application type */
  MQPID     ProcessId;      /* Process identifier */
  MQTID     ThreadId;       /* Thread identifier */
};
```

MQAXP - API exit parameter

This table summarizes the fields in the structure.

Table 31. Fields in MQAXP

Field	Description
StrucId	Structure identifier
Version	Structure version number
ExitId	Type of exit
ExitReason	Reason for invoking exit
ExitResponse	Response from exit
ExitResponse2	Secondary response from exit
Feedback	Feedback code
APICallerType	API caller type
ExitUserArea	Exit user area
ExitData	Exit data
ExitInfoName	Exit information name
ExitPDArea	Problem determination area
QMgrName	Name of local queue manager
ExitChainAreaPtr	Address of first chain area
Hconfig	Configuration handle
Function	API function identifier

The MQAXP structure describes the information that is passed to an API exit.

Fields

The MQAXP structure contains thes fields:

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQAXP_STRUC_ID
Identifier for API exit parameter structure.

For the C programming language, the constant MQAXP_STRUC_ID_ARRAY is also defined; this has the same value as MQAXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

The value is:

MQAXP_VERSION_1
Version-1 API exit parameter structure.

The following constant specifies the version number of the current version:

MQAXP_CURRENT_VERSION
Current version of API exit parameter structure.

Note: When a new version of the MQAXP structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG)
Type of exit.

This indicates the type of exit being called.

The value is:

MQXT_API_EXIT
API exit.

This is an input field to the exit.

ExitReason (MQLONG)
Reason for invoking exit.

This indicates the reason why the exit is being called. Possible values are:

MQXR_CONNECTION
Connection level processing.

The exit is invoked with this value twice for each connection:

- Before the MQCONN or MQCONNX call, so that the exit can perform connection-level initialization. The Function field has the value MQXF_INIT in this case.

  The MQXF_INIT exit function should be used for general initialization of the exit suite, and the MQXF_CONN or MQXF_CONNX exit functions should be used specifically for processing the MQCONN or MQCONNX calls.
- After the MQDISC call, so that the exit can perform connection-level termination. The Function field has the value MQXF_TERM in this case.

  The MQXF_TERM exit function should be used for general termination of the exit suite, and the MQXF_DISC exit function should be used specifically for processing the MQDISC call.

MQXR_BEFORE
Before API execution.

The Function field can have any of the MQXF_* values other than MQXF_INIT or MQXF_TERM.

For the MQGET call, this value occurs with the:

- MQXF_GET exit function before API execution
- MQXF_DATA_CONV_ON_GET exit function after API execution but before data conversion

MQXR_AFTER
After API execution.

The Function field can have any of the MQXF_* values other than MQXF_INIT, MQXF_TERM, or MQXF_DATA_CONV_ON_GET.

For the MQGET call, this value occurs with the:

- MQXF_GET exit function after both API execution and data conversion have been completed

This is an input field to the exit.

ExitResponse (MQLONG)
Response from exit.

This is set by the exit function to indicate the outcome of the processing performed by the exit. It must be one of the following:

MQXCC_OK
Exit completed successfully.

This value can be set by all MQXR_* exit functions. The ExitResponse2 field must be set by the exit function to indicate how processing should continue.

Note: Returning MQXCC_OK does not imply that the completion code for the API call is MQCC_OK, or that the reason code is MQRC_NONE.

MQXCC_FAILED
Exit failed.

This value can be set by all MQXR_* exit functions. It causes the queue manager to set the completion code for the API call to MQCC_FAILED, and the reason code to one of the following values:

```
Exit function   Reason code set by queue manager
---------------------------------------------
MQXF_INIT       MQRC_API_EXIT_INIT_ERROR
MQXF_TERM       MQRC_API_EXIT_TERM_ERROR
All others      MQRC_API_EXIT_ERROR
```

However, the values set by the queue manager can be altered by an exit function later in the chain. The ExitResponse2 field is ignored; the queue manager continues processing as though MQXR2_SUPPRESS_CHAIN had been returned:

- For an MQXR_BEFORE exit function, processing continues with the MQXR_AFTER exit function that matches this MQXR_BEFORE exit function (that is, all intervening MQXR_BEFORE and MQXR_AFTER exit functions, plus the API call itself, are skipped).
- For an MQXR_AFTER exit function, processing continues with the next MQXR_AFTER exit function in the chain.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

If an MQXR_BEFORE exit function returns this value, the queue manager sets the completion code for the API call to MQCC_FAILED, the reason code to MQRC_SUPPRESSED_BY_EXIT, and the API call is skipped. If returned by the MQXF_DATA_CONV_ON_GET exit function, data conversion is skipped.

The ExitResponse2 field must be set by the exit function to indicate whether the remaining MQXR_BEFORE exit functions and their matching MQXR_AFTER exit functions should be invoked. Any of

these exit functions can alter the completion code and reason code of the API call that were set by the queue manager.

If an MQXR_AFTER or MQXR_CONNECTION exit function returns this value, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

MQXCC_SKIP_FUNCTION
Skip function.

This is the same as MQXCC_SUPPRESS_FUNCTION, except the exit function can set the completion code and reason code of the API call.

MQXCC_SUPPRESS_EXIT
Suppress exit.

If an MQXR_BEFORE or MQXR_AFTER exit function returns this value, the queue manager deregisters immediately all of the exit functions belonging to this exit suite. The only exception is the MQXF_TERM exit function, which will be invoked at termination of the connection if registered when MQXCC_SUPPRESS_EXIT is returned. Note that if an MQXR_BEFORE exit function returns this value, the matching MQXR_AFTER exit function will not be invoked after the API call, since that exit function will no longer be registered.

The ExitResponse2 field must be set by the exit function to indicate whether the remaining MQXR_BEFORE exit functions and their matching MQXR_AFTER exit functions should be invoked.

If an MQXR_CONNECTION exit function returns this value, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

If the exit function sets ExitResponse to a value that is not valid, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

On entry to the exit function, ExitResponse has the value MQXCC_OK.

This is an output field from the exit.

ExitResponse2 (MQLONG)
Secondary response from exit.

This is the secondary exit response code that can be set by an MQXR_BEFORE exit function to provide additional information to the queue manager. If set by an MQXR_AFTER or MQXR_CONNECTION exit function, the value is ignored. The value must be one of the following:

MQXR2_DEFAULT_CONTINUATION
Default continuation.

Continuation with the next exit function in the chain depends on the value of the ExitResponse field:

- If ExitResponse is MQXCC_OK or MQXCC_SUPPRESS_EXIT, the next MQXR_BEFORE exit function in the chain is invoked.
- If ExitResponse is MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION, no further MQXR_BEFORE exit functions are invoked for this particular API call.

MQXR2_CONTINUE_CHAIN
Continue with next MQXR_BEFORE exit function in chain.

MQXR2_SUPPRESS_CHAIN
Skip remaining MQXR_BEFORE exit functions in chain.

All subsequent MQXR_BEFORE exit functions in the chain, and their matching MQXR_AFTER exit functions, are skipped for this particular API call. The MQXR_AFTER exit functions that match the current exit function and earlier MQXR_BEFORE exit functions are not skipped.

If the exit function sets ExitResponse2 to a value that is not valid, the queue manager continues processing as though the exit had returned MQXR2_DEFAULT_CONTINUATION.

This is an output field from the exit.

Feedback (MQLONG)
Feedback.

This is a field that allows the exit functions belonging to an exit suite to communicate feedback codes both to each other, and to exit functions belonging to other exit suites. The field is initialized to MQFB_NONE before the first invocation of the first exit function in the first exit suite (the MQXF_INIT exit function), and thereafter any changes made to this field by exit functions are preserved across the invocations of the exit functions.

This is an input/output field to the exit.

APICallerType (MQLONG)
API caller type.

This indicates the type of program that issued the API call that caused the exit function to be invoked.

The value is one of the following:

MQXACT_EXTERNAL
Caller is external to the queue manager.

MQXACT_INTERNAL
Caller is internal to the queue manager.

This is an input field to the exit.

ExitUserArea (MQBYTE16)
Exit user area.

This is a field that allows exit functions belonging to the same exit suite to share data with each other, but not with other exit suites. The field is initialized to MQXUA_NONE (binary zero) before the first invocation of the first exit function in the exit suite (the MQXF_INIT exit function), and thereafter any changes made to this field by exit functions are preserved across the invocations of the exit functions. The queue manager resets the field to MQXUA_NONE when control returns from the MQXF_TERM exit function to the queue manager.

The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field. For the C programming language, the constant MQXUA_NONE_ARRAY is

also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32)
Exit data.

On input to each exit function, this field is set to the character data associated with the definition of the exit suite to which the exit function belongs. If no value has been defined for that data, ExitData is blank.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This is an input field to the exit.

ExitInfoName (MQCHAR48)
Exit information name.

This is a name that is used to identify the exit suite to which the exit function belongs.

The length of this field is given by MQ_EXIT_INFO_NAME_LENGTH. This is an input field to the exit.

ExitPDArea (MQBYTE48)
Problem determination area.

This is a field that is available for the exit to use, to assist with problem determination. The field is initialized to MQXPDA_NONE (binary zero) before each invocation of the exit function. The exit function can set this field to any value it chooses. When the exit returns control to the queue manager, the contents of ExitPDArea are written to the trace file, if tracing is active.

The following value is defined:

MQXPDA_NONE
No problem-determination information.

The value is binary zero for the length of the field. For the C programming language, the constant MQXPDA_NONE_ARRAY is also defined; this has the same value as MQXPDA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_PD_AREA_LENGTH. This is an input/output field to the exit.

QMgrName (MQCHAR48)
Name of local queue manager.

This is the name of the queue manager that invoked the exit function. QMgrName is never blank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input field to the exit.

ExitChainAreaPtr (PMQACH)
Address of first MQACH structure in chain.

The exit chain area allows exit functions belonging to one exit suite to share data with exit functions belonging to another exit suite. The exit chain area is a chain of MQACH structures that is made available to all exit functions. The address of the first MQACH structure in the chain is

passed to each exit function in the ExitChainAreaPtr field. The exit function can scan the chain, and examine or alter the data contained within it. However, this should be done only with the prior agreement of the owner of the data.

If there is no current exit chain area, ExitChainAreaPtr is the NULL pointer. An exit function can at any time create an MQACH structure in storage obtained dynamically (for example, by using the C function malloc), and add it to the chain. The exit suite which creates an MQACH is responsible for freeing the storage associated with the MQACH before the exit suite terminates.

If data is to be shared between different exit functions belonging to the same exit suite, but that data is not to be made available to other exit suites, the ExitUserArea field should be used in preference to ExitChainAreaPtr.

This is an input/output field to the exit.

Hconfig (MQHCONFIG)
Configuration handle.

This handle represents the set of exit functions that belong to the exit suite whose name is given by the ExitInfoName field. The queue manager generates a new configuration handle when the MQXF_INIT exit function is invoked, and passes that handle to the other exit functions that belong to the exit suite. This handle must be specified on the MQXEP call in order to register the entry point for an exit function.

This is an input field to the exit.

Function (MQLONG)
API function identifier.

This is the identifier of the API call that is about to be executed (when ExitReason has the value MQXR_BEFORE), or the API call that has just been executed (when ExitReason has the value MQXR_AFTER). If ExitReason has the value MQXR_CONNECTION, Function indicates whether the exit should perform initialization or termination. The value is one of the following:

MQXF_INIT
Initialization of exit suite.

MQXF_TERM
Termination of exit suite.

MQXF_CONN
MQCONN call.

MQXF_DISC
MQDISC call.

MQXF_OPEN
MQOPEN call.

MQXF_CLOSE
MQCLOSE call.

MQXF_PUT
MQPUT call.

MQXF_GET
MQGET call.

MQXF_DATA_CONV_ON_GET
Data conversion on MQGET call.

MQXF_INQ
MQINQ call.

MQXF_SET
MQSET call.

MQXF_CMIT
MQCMIT call.

MQXF_BACK
MQBACK call.

This is an input field to the exit.

C declaration

```
typedef struct tagMQAXP MQAXP;
struct tagMQAXP
{
  MQCHAR4   StrucId;    /* Structure identifier */
  MQLONG    Version;    /* Structure version number */
  MQLONG    ExitId;     /* Type of exit */
  MQLONG    ExitReason;     /* Reason for invoking exit */
  MQLONG    ExitResponse;   /* Response from exit */
  MQLONG    ExitResponse2;  /* Secondary response from exit */
  MQLONG    Feedback;     /* Feedback */
  MQLONG    APICallerType;  /* API caller type */
  MQBYTE16  ExitUserArea;   /* Exit user area */
  MQCHAR32  ExitData;     /* Exit data */
  MQCHAR48  ExitInfoName;   /* Exit information name */
  MQBYTE48  ExitPDArea;     /* Problem determination area */
  MQCHAR48  QMgrName;     /* Name of local queue manager */
  PMQACH    ExitChainAreaPtr;   /* Address of first MQACH structure in chain */
  MQHCONFIG Hconfig;    /* Configuration handle */
  MQLONG    Function;     /* API function identifier */
};
```

MQXEP - Register entry point

This call is used by an exit function to register the entry points of other exit functions in the exit suite. This is usually done by the MQ_INIT_EXIT function, but can be done by any exit function in the exit suite.

The MQXEP call is also used to deregister entry points. This is usually done by the MQ_TERM_EXIT function, but can be done by any exit function in the exit suite.

Syntax

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, Reserved, pCompCode, pReason)
```

Parameters

The MQXEP call has these parameters:

Hconfig (MQHCONFIG) - input
Configuration handle.

This handle represents the exit suite to which the current exit function belongs. The queue manager generates this configuration handle when the MQ_INIT_EXIT function is invoked, and uses the Hconfig field in the MQAXP structure to pass the handle to each exit function in the exit suite.

ExitReason (MQLONG) - input
Exit reason.

This specifies when to call the entry point being registered or deregistered. It must be one of the following:

MQXR_CONNECTION
Connection level processing.

The Function parameter must have the value MQXF_INIT or MQXF_TERM.

MQXR_BEFORE
Before API execution.

The Function parameter can have any of the MQXF_* values other than MQXF_INIT or MQXF_TERM.

MQXR_AFTER
After API execution.

The Function parameter can have any of the MQXF_* values other than MQXF_INIT, MQXF_TERM, or MQXF_DATA_CONV_ON_GET.

Function (MQLONG) - input
Function identifier.

This specifies the API call for which the entry point is being registered or deregistered. It must be one of the following:

MQXF_INIT
Initialization of exit suite.

MQXF_TERM
Termination of exit suite.

MQXF_CONN
MQCONN call.

MQXF_DISC
MQDISC call.

MQXF_OPEN
MQOPEN call.

MQXF_CLOSE
MQCLOSE call.

MQXF_PUT
MQPUT call.

MQXF_GET
MQGET call.

MQXF_DATA_CONV_ON_GET
Data conversion on MQGET call.

MQXF_INQ
MQINQ call.

MQXF_SET
MQSET call.

MQXF_CMIT
MQCMIT call.

MQXF_BACK
MQBACK call.

If the MQXEP call is used more than once to register different entry points for a particular combination of Function and ExitReason, the last call made provides the entry point that is used.

EntryPoint (PMQFUNC) - input
Exit function entry point.

This is the address of the entry point being registered.

If the value specified is the null pointer, it indicates either that the exit function is not provided, or that a previously-registered exit function is being deregistered. The null pointer is assumed for entry points which are not defined using MQXEP.

Reserved (MQPTR) - input
Reserved.

This is a reserved parameter. The value specified must be the null pointer.

pCompCode (PMQLONG) - output
Completion code.

The value returned is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

pReason (PMQLONG) - output
Reason code qualifying pCompCode.

If CompCode is MQCC_OK:

MQRC_NONE (0, X'000')
No reason to report.

If CompCode is MQCC_FAILED:

MQRC_EXIT_REASON_ERROR (2377, X'949')
Exit reason not valid.

MQRC_FUNCTION_ERROR (2281, X'8E9')
Function identifier not valid.

MQRC_HCONFIG_ERROR (2280, X'8E8')
Configuration handle not valid.

MQRC_RESERVED_VALUE_ERROR (2378, X'94A')
Reserved value not valid.

MQRC_RESOURCE_PROBLEM (2102, X'836')
Insufficient system resources available.

MQRC_UNEXPECTED_ERROR (2195, X'893')
Unexpected error occurred.

For more information on these reason codes, see the WebSphere MQ Application Programming Reference.

C invocation

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, Reserved, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG Hconfig;  /* Configuration handle */
MQLONG ExitReason;  /* Exit reason */
MQLONG Function;  /* Function identifier */
PMQFUNC EntryPoint;   /* Exit function entry point */
MQPTR Reserved;   /* Reserved */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

MQ_BACK_EXIT - Back out changes

Exit providers can supply an MQ_BACK_EXIT function to intercept the MQBACK call.

Syntax

```
MQ_BACK_EXIT (pExitParms, pExitContext, pHconn, pCompCode, pReason)
```

Parameters

The MQ_BACK_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP pExitParms;  /* Exit parameter structure */
PMQAXC pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

MQ_CLOSE_EXIT - Close object

Exit providers can supply an MQ_CLOSE_EXIT function to intercept the MQCLOSE call.

Syntax

```
MQ_CLOSE_EXIT (pExitParms, pExitContext, pHconn, ppHobj, pOptions, pCompCode, pReason)
```

Parameters

The MQ_CLOSE_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

ppHobj (PPMQHOBJ) - input/output
Object handle.

pOptions (PMQLONG) - input/output
Options that control the action of MQCLOSE.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHobj, &Options, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;  /* Exit parameter structure */
PMQAXC   pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PPMQHOBJ ppHobj;  /* Object handle */
PMQLONG  pOptions;  /* Options that control the action of MQCLOSE */
PMQLONG  pCompCode;   /* Completion code */
PMQLONG  pReason;   /* Reason code qualifying CompCode */
```

MQ_CMIT_EXIT - Commit changes

Exit providers can supply an MQ_CMIT_EXIT function to intercept the MQCMIT call.

Syntax

```
MQ_CMIT_EXIT (pExitParms, pExitContext, pHconn, pCompCode, pReason)
```

Parameters

The MQ_CMIT_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_CMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;  /* Exit parameter structure */
PMQAXC   pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PMQLONG  pCompCode;   /* Completion code */
PMQLONG  pReason;   /* Reason code qualifying CompCode */
```

MQ_CONNX_EXIT - Connect queue manager (extended)

Exit providers can supply an MQ_CONNX_EXIT function to intercept the MQCONN and MQCONNX calls.

Syntax

```
MQ_CONNX_EXIT (pExitParms, pExitContext, pQMgrName, ppConnectOpts, ppHconn,
         pCompCode, pReason)
```

Parameters

The MQ_CONNX_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pQMgrName (PMQCHAR48) - input/output
Name of queue manager.

ppConnectOpts (PPMQCNO) - input/output
Options that control the action of MQCONNX.

ppHconn (PPMQHCONN) - input/output
Connection handle.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

Usage notes

1. The MQ_CONNX_EXIT function interface described here is used for both the MQCONN call and the MQCONNX call. However, WebSphere MQ for z/VSE only supports the MQXF_CONN API exit point.
2. When a message channel agent (MCA) responds to an inbound client connection, the MCA can issue a number of MQ calls before the client state is fully known. These MQ calls result in the API exit functions being invoked with the MQAXC structure containing data relating to the MCA, and not to the client (for example, user identifier and connection name). However, once the client state is fully known, subsequent MQ calls result in the API exit functions being invoked with the appropriate client data in the MQAXC structure.

3. All MQXR_BEFORE exit functions are invoked before any parameter validation is performed by the queue manager. The parameters may therefore be invalid (including invalid pointers for the addresses of parameters).

   The MQ_CONNX_EXIT function is invoked before any authorization checks are performed by the queue manager.
4. The exit function must not change the name of the queue manager specified on the MQCONN or MQCONNX call. If the name is changed by the exit function, the results are undefined.
5. An MQXR_BEFORE exit function for the MQ_CONNX_EXIT cannot issue MQ calls other than MQXEP.

C invocation

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, QMgrName, &pConnectOpts, &pHconn,
               &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP pExitParms;  /* Exit parameter structure */
PMQAXC pExitContext;  /* Exit context structure */
PMQCHAR48 pQMgrName;  /* Name of queue manager */
PPMQCNO ppConnectOpts;  /* Options that control the action of MQCONNX */
PPMQHCONN ppHconn;  /* Connection handle */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

MQ_DISC_EXIT - Disconnect queue manager

Exit providers can supply an MQ_DISC_EXIT function to intercept the MQDISC call.

Syntax

```
MQ_DISC_EXIT (pExitParms, pExitContext, ppHconn, pCompCode, pReason)
```

Parameters

The MQ_DISC_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

ppHconn (PPMQHCONN) - input/output
Connection handle.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP pExitParms;  /* Exit parameter structure */
PMQAXC pExitContext;  /* Exit context structure */
PPMQHCONN ppHconn;  /* Connection handle */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

MQ_GET_EXIT - Get message

Exit providers can supply an MQ_GET_EXIT function to intercept the MQGET call. The same exit function interface is used for the MQXF_DATA_CONV_ON_GET exit function.

Syntax

```
MQ_GET_EXIT (pExitParms, pExitContext, pHconn, pHobj, ppMsgDesc, ppGetMsgOpts,
             pBufferLength, ppBuffer, ppDataLength, pCompCode, pReason)
```

Parameters

The MQ_GET_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pHobj (PMQHOBJ) - input/output
Object handle.

ppMsgDesc (PPMQMD) - input/output
Message descriptor.

ppGetMsgOpts (PPMQGMO) - input/output
Options that control the action of MQGET.

pBufferLength (PMQLONG) - input/output
Length in bytes of the ppBuffer area.

ppBuffer (PPMQVOID) - input/output
Area to contain the message data.

ppDataLength (PPMQLONG) - input/output
Length of the message.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the MQXF_DATA_CONV_ON_GET exit function. However, separate entry points are defined for these two exit functions, so to intercept both the MQXEP call must be used twice - once with function identifier MQXF_GET, and again with MQXF_DATA_CONV_ON_GET.

   Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the

Function field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

2. There is no MQXR_AFTER exit function for MQXF_DATA_CONV_ON_GET; the MQXR_AFTER exit function for MQXF_GET provides the required capability for exit processing after data conversion.

C invocation

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc, &pGetMsgOpts,
       &BufferLength, &pBuffer, &pDataLength, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;  /* Exit parameter structure */
PMQAXC   pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PMQHOBJ  pHobj;   /* Object handle */
PPMQMD   ppMsgDesc;   /* Message descriptor */
PPMQGMO  ppGetMsgOpts;  /* Options that control the action of MQGET */
PMQLONG  pBufferLength; /* Length in bytes of the pBuffer area */
PPMQVOID ppBuffer;  /* Area to contain the message data */
PPMQLONG ppDataLength;  /* Length of the message */
PMQLONG  pCompCode;   /* Completion code */
PMQLONG  pReason;   /* Reason code qualifying CompCode */
```

MQ_INIT_EXIT - Initialize exit environment

Exit providers can supply an MQ_INIT_EXIT function to perform connection-level initialization.

Syntax

```
MQ_INIT_EXIT (pExitParms, pExitContext, pCompCode, pReason)
```

Parameters

The MQ_INIT_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

Usage notes

1. The MQ_INIT_EXIT function can issue the MQXEP call to register the addresses of the exit functions for the particular MQ calls to be intercepted. It is not necessary to intercept all MQ calls, or to intercept both MQXR_BEFORE and MQXR_AFTER calls. For example, an exit suite could choose to intercept only the MQXR_BEFORE call of MQPUT.
2. Storage that is to be used by exit functions in the exit suite can be acquired by the MQ_INIT_EXIT function. Alternatively, exit functions can acquire storage when they are invoked, as and when needed. However, all storage should be

freed before the exit suite is terminated; the MQ_TERM_EXIT function can free the storage, or an exit function invoked earlier.

3. If MQ_INIT_EXIT returns MQXCC_FAILED in the ExitResponse field of MQAXP, or fails in some other way, the MQCONN or MQCONNX call that caused MQ_INIT_EXIT to be invoked also fails, with the CompCode and Reason parameters set to appropriate values.
4. An MQ_INIT_EXIT function cannot issue MQ calls other than MQXEP.

C invocation

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP pExitParms;  /* Exit parameter structure */
PMQAXC pExitContext;  /* Exit context structure */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

MQ_INQ_EXIT - Inquire object attributes

Exit providers can supply an MQ_INQ_EXIT function to intercept the MQINQ call.

Syntax

```
MQ_INQ_EXIT (pExitParms, pExitContext, pHconn, pHobj, pSelectorCount, ppSelectors,
       pIntAttrCount, ppIntAttrs, pCharAttrLength, ppCharAttrs, pCompCode,
       pReason)
```

Parameters

The MQ_INQ_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pHobj (PMQHOBJ) - input/output
Object handle.

pSelectorCount (PMQLONG) - input/output
Count of selectors.

ppSelectors (PPMQLONG) - input/output
Array of attribute selectors.

pIntAttrCount (PMQLONG) - input/output
Count of integer attributes.

ppIntAttrs (PPMQLONG) - input/output
Array of integer attributes.

pCharAttrLength (PMQLONG) - input/output
Length of character attributes buffer.

ppCharAttrs (PPMQCHAR) - input/output
Character attributes.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount, &pSelectors,
       &IntAttrCount, &pIntAttrs, &CharAttrLength, &pCharAttrs, &CompCode,
       &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;    /* Exit parameter structure */
PMQAXC   pExitContext;    /* Exit context structure */
PMQHCONN pHconn;    /* Connection handle */
PMQHOBJ  pHobj;     /* Object handle */
PMQLONG  pSelectorCount;  /* Count of selectors */
PPMQLONG ppSelectors;     /* Array of attribute selectors */
PMQLONG  pIntAttrCount;   /* Count of integer attributes */
PPMQLONG ppIntAttrs;    /* Array of integer attributes */
PMQLONG  pCharAttrLength;   /* Length of character attributes buffer */
PPMQCHAR ppCharAttrs;     /* Character attributes */
PMQLONG  pCompCode;     /* Completion code */
PMQLONG  pReason;     /* Reason code qualifying CompCode */
```

MQ_OPEN_EXIT - Open object

Exit providers can supply an MQ_OPEN_EXIT function to intercept the MQOPEN call.

Syntax

```
MQ_OPEN_EXIT (pExitParms, pExitContext, pHconn, ppObjDesc, pOptions, ppHobj,
        pCompCode, pReason)
```

Parameters

The MQ_OPEN_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

ppObjDesc (PPMQOD) - input/output
Object descriptor.

pOptions (PMQLONG) - input/output
Options that control the action of MQ_OPEN_EXIT.

ppHobj (PPMQHOBJ) - input/output
Object handle.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &Options, &pHobj,
        &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;  /* Exit parameter structure */
PMQAXC   pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PPMQOD   ppObjDesc;   /* Object descriptor */
PMQLONG  pOptions;  /* Options that control the action of MQ_OPEN_EXIT */
PPMQHOBJ ppHobj;  /* Object handle */
PMQLONG  pCompCode;   /* Completion code */
PMQLONG  pReason;   /* Reason code qualifying CompCode */
```

MQ_PUT_EXIT - Put message

Exit providers can supply an MQ_PUT_EXIT function to intercept the MQPUT call.

The MQ_PUT_EXIT function can also be used to intercept the MQPUT1 call.

Syntax

```
MQ_PUT_EXIT (pExitParms, pExitContext, pHconn, pHobj, ppMsgDesc, ppPutMsgOpts,
       pBufferLength, ppBuffer, pCompCode, pReason)
```

Parameters

The MQ_PUT_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pHobj (PMQHOBJ) - input/output
Object handle.

ppMsgDesc (PPMQMD) - input/output
Message descriptor.

ppPutMsgOpts (PPMQPMO) - input/output
Options that control the action of MQPUT.

pBufferLength (PMQLONG) - input/output
Length of the message in pBuffer.

ppBuffer (PPMQVOID) - input/output
Message data.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc, &pPutMsgOpts,
       &BufferLength, &pBuffer, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;  /* Exit parameter structure */
PMQAXC   pExitContext;  /* Exit context structure */
PMQHCONN pHconn;  /* Connection handle */
PMQHOBJ  pHobj;   /* Object handle */
PPMQMD   ppMsgDesc;   /* Message descriptor */
PPMQPMO  ppPutMsgOpts;  /* Options that control the action of MQPUT */
PMQLONG  pBufferLength; /* Length of the message in pBuffer */
PPMQVOID ppBuffer;  /* Message data */
PMQLONG  pCompCode;   /* Completion code */
PMQLONG  pReason;   /* Reason code qualifying CompCode */
```

MQ_SET_EXIT - Set object attributes

Exit providers can supply an MQ_SET_EXIT function to intercept the MQSET call.

Syntax

```
MQ_SET_EXIT (pExitParms, pExitContext, pHconn, pHobj, pSelectorCount,
ppSelectors, pIntAttrCount, ppIntAttrs, pCharAttrLength, ppCharAttrs,
pCompCode, pReason)
```

Parameters

The MQ_SET_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input/output
Connection handle.

pHobj (PMQHOBJ) - input/output
Object handle.

pSelectorCount (PMQLONG) - input/output
Count of selectors.

ppSelectors (PPMQLONG) - input/output
Array of attribute selectors.

pIntAttrCount (PMQLONG) - input/output
Count of integer attributes.

ppIntAttrs (PPMQLONG) - input/output
Array of integer attributes.

pCharAttrLength (PMQLONG) - input/output
Length of character attributes buffer.

ppCharAttrs (PPMQCHAR) - input/output
Character attributes.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

C invocation

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount, &pSelectors,
       &IntAttrCount, &pIntAttrs, &CharAttrLength, &pCharAttrs, &CompCode,
       &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP   pExitParms;    /* Exit parameter structure */
PMQAXC   pExitContext;    /* Exit context structure */
PMQHCONN pHconn;    /* Connection handle */
PMQHOBJ  pHobj;     /* Object handle */
PMQLONG  pSelectorCount;  /* Count of selectors */
PPMQLONG ppSelectors;     /* Array of attribute selectors */
PMQLONG  pIntAttrCount;   /* Count of integer attributes */
PPMQLONG ppIntAttrs;    /* Array of integer attributes */
PMQLONG  pCharAttrLength;   /* Length of character attributes buffer */
PPMQCHAR ppCharAttrs;     /* Character attributes */
PMQLONG  pCompCode;     /* Completion code */
PMQLONG  pReason;     /* Reason code qualifying CompCode */
```

MQ_TERM_EXIT - Terminate exit environment

Exit providers can supply an MQ_INIT_EXIT function to perform connection-level termination.

Syntax

```
MQ_TERM_EXIT (pExitParms, pExitContext, pCompCode, pReason)
```

Parameters

The MQ_TERM_EXIT call has these parameters:

pExitParms (PMQAXP) - input/output
Exit parameter structure.

pExitContext (PMQAXC) - input/output
Exit context structure.

pCompCode (PMQLONG) - input/output
Completion code.

pReason (PMQLONG) - input/output
Reason code qualifying pCompCode.

Usage notes

1. The MQ_TERM_EXIT function is optional. It is not necessary for an exit suite to register a termination exit if there is no termination processing to be done.

   If functions belonging to the exit suite acquire resources during the connection, an MQ_TERM_EXIT function is a convenient point at which to free those resources, for example, freeing storage obtained dynamically.
2. If an MQ_TERM_EXIT function is registered when the MQDISC call is issued, the exit function is invoked after all of the MQDISC exit functions have been invoked.
3. If MQ_TERM_EXIT returns MQXCC_FAILED in the ExitResponse field of MQAXP, or fails in some other way, the MQDISC call that caused MQ_TERM_EXIT to be invoked also fails, with the CompCode and Reason parameters set to appropriate values.

C invocation

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP  pExitParms;   /* Exit parameter structure */
PMQAXC  pExitContext;   /* Exit context structure */
PMQLONG pCompCode;  /* Completion code */
PMQLONG pReason;  /* Reason code qualifying CompCode */
```

Appendix A. CICS control table definitions

This appendix contains various sample entries for the CICS control tables.

Sample file control table entries

FCT macro definitions are only necessary for WebSphere MQ for z/VSE running under CICS for z/VSE. Matching file definitions under CICS TS are part of the CICS CSD, and are defined using the DEFINE FILE CSD command. Consequently, the following sample is for MQ under CICS for z/VSE only.

```
*------------------------------------------------------------------*
* Licensed Materials - Property of IBM                             *
* 5655-U97                                                         *
* Copyright IBM Corp. 2008                                         *
*                                                                  *
* US Government Users Restricted Rights - Use, duplication or      *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.  *
*------------------------------------------------------------------*
*
*------------------------------------------------------------------*
*         Start of  WebSphere MQ VSAM cluster definitions          *
*                                                                  *
*  For performance reasons entries may be modified to add LSRPOOL  *
*  explicit specifications.                                        *
*------------------------------------------------------------------*
*
* system setup file
MQFSSET  DFHFCT TYPE=DATASET,DATASET=MQFSSET,                        *
               ACCMETH=VSAM,                                        *
               SERVREQ=(READ,BROWSE),                               *
               LOG=NO,                                              *
               RSL=PUBLIC,                                          *
               STRNO=5,                                             *
               RECFORM=(FIXED,BLOCKED)
* configuration file
MQFCNFG  DFHFCT TYPE=DATASET,DATASET=MQFCNFG,                        *
               ACCMETH=VSAM,                                        *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),             *
               LOG=YES,                                             *
               RSL=PUBLIC,                                          *
               STRNO=20,                                            *
               RECFORM=(FIXED,BLOCKED)
*--reorganization file
MQFREOR  DFHFCT TYPE=DATASET,DATASET=MQFREOR,                        *
               ACCMETH=VSAM,                                        *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),             *
               RSL=PUBLIC,                                          *
               LOG=NO,                                              *
               STRNO=16,                                            *
               RECFORM=(VARIABLE,BLOCKED)
*--MQSC and MQSC command and reply queues
MQFADMN  DFHFCT TYPE=DATASET,DATASET=MQFADMN,                        *
               ACCMETH=VSAM,                                        *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),             *
               RSL=PUBLIC,                                          *
               LOG=YES,                                             *
               STRNO=16,                                            *
               RECFORM=(VARIABLE,BLOCKED)
*--WebSphere MQ Explorer model and reply queues
MQFDEFS  DFHFCT TYPE=DATASET,DATASET=MQFDEFS,                        *
               ACCMETH=VSAM,                                        *
```

```
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                RSL=PUBLIC,                                            *
                LOG=YES,                                               *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
*--example of queues (input followed by output)

MQFI001  DFHFCT TYPE=DATASET,DATASET=MQFI001,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                RSL=PUBLIC,                                            *
                LOG=YES,                                               *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
MQFO001  DFHFCT TYPE=DATASET,DATASET=MQFO001,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                LOG=YES,                                               *
                RSL=PUBLIC,                                            *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
MQFI002  DFHFCT TYPE=DATASET,DATASET=MQFI002,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                RSL=PUBLIC,                                            *
                LOG=YES,                                               *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
MQFO002  DFHFCT TYPE=DATASET,DATASET=MQFO002,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                LOG=YES,                                               *
                RSL=PUBLIC,                                            *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
MQFI003  DFHFCT TYPE=DATASET,DATASET=MQFI003,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                LOG=YES,                                               *
                RSL=PUBLIC,                                            *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
MQFO003  DFHFCT TYPE=DATASET,DATASET=MQFO003,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                LOG=YES,                                               *
                RSL=PUBLIC,                                            *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
*--WebSphere MQ Accounting Messages queue
MQFACCT  DFHFCT TYPE=DATASET,DATASET=MQFACCT,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                RSL=PUBLIC,                                            *
                LOG=YES,                                               *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
*--WebSphere MQ Statistics Messages queue
MQFSTAT  DFHFCT TYPE=DATASET,DATASET=MQFSTAT,                          *
                ACCMETH=VSAM,                                          *
                SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
                RSL=PUBLIC,                                            *
                LOG=YES,                                               *
                STRNO=16,                                              *
                RECFORM=(VARIABLE,BLOCKED)
*--SYSTEM DEFINITIONS
MQFLOG   DFHFCT TYPE=DATASET,DATASET=MQFLOG,                           *
```

```
               ACCMETH=VSAM,                                          *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
               LOG=YES,                                               *
               RSL=PUBLIC,                                            *
               STRNO=16,                                              *
               RECFORM=(VARIABLE,BLOCKED)
MQFERR   DFHFCT TYPE=DATASET,DATASET=MQFERR,                          *
               ACCMETH=VSAM,                                          *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
               LOG=YES,                                               *
               RSL=PUBLIC,                                            *
               STRNO=16,                                              *
               RECFORM=(VARIABLE,BLOCKED)
MQFMON   DFHFCT TYPE=DATASET,DATASET=MQFMON,                          *
               ACCMETH=VSAM,                                          *
               SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),               *
               LOG=NO,                                                *
               RSL=PUBLIC,                                            *
               STRNO=16,                                              *
               RECFORM=(VARIABLE,BLOCKED)
*---------------------------------------------------------------------*
*             End of WebSphere MQ VSAM cluster definitions            *
*---------------------------------------------------------------------*
```

The following sample definitions are also provided for the Programmable Command Formats (PCF) and WebSphere MQ Command (MQSC) features:

```
*--PCF SYSTEM DEFINITIONS
* MQFACMD  DFHFCT TYPE=DATASET,DATASET=MQFACMD,
*                 ACCMETH=VSAM,
*                 SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),
*                 LOG=YES,
*                 RSL=PUBLIC,
*                 STRNO=16,
*                 RECFORM=(VARIABLE,BLOCKED)
* MQFARPY  DFHFCT TYPE=DATASET,DATASET=MQFARPY,
*                 ACCMETH=VSAM,
*                 SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),
*                 LOG=YES,
*                 RSL=PUBLIC,
*                 STRNO=16,
*                 RECFORM=(VARIABLE,BLOCKED)
```

Similarly, the following sample definitions are also provided for the Instrumentation Events feature:

```
*--INSTRUMENTATION EVENT DEFINITIONS
* MQFIEQE  DFHFCT TYPE=DATASET,DATASET=MQFIEQE,
*                 ACCMETH=VSAM,
*                 SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),
*                 LOG=YES,
*                 RSL=PUBLIC,
*                 STRNO=16,
*                 RECFORM=(VARIABLE,BLOCKED)
* MQFIECE  DFHFCT TYPE=DATASET,DATASET=MQFIECE,
*                 ACCMETH=VSAM,
*                 SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),
*                 LOG=YES,
*                 RSL=PUBLIC,
*                 STRNO=16,
*                 RECFORM=(VARIABLE,BLOCKED)
* MQFIEPE  DFHFCT TYPE=DATASET,DATASET=MQFIEPE,
*                 ACCMETH=VSAM,
*                 SERVREQ=(READ,UPDATE,ADD,BROWSE,DELETE),
*                 LOG=YES,
```

```
*               RSL=PUBLIC,
*               STRNO=16,
*               RECFORM=(VARIABLE,BLOCKED)
```

Sample destination control table entry

Destination control table entry MQER is required in order for WebSphere MQ system messages to be logged to the SYSTEM.LOG queue.

In the event that security is installed for the WebSphere MQ subsystem, the DCT entry may need to be modified to include a "USERID=user" parameter. For information on this requirement, refer to “Changing the MQER TDQ definition” on page 21.

Similarly, destination control table entry MQXP is required for the queue manager to manage expired messages.

Once again, if security is installed, the DCT entry may need to be modified according to the instructions described in section “Changing the MQXP TDQ definition” on page 22.

```
*-------------------------------------------------------------------*
* Licensed Materials - Property of IBM                              *
*                                                                   *
* 5655-U97                                                          *
* Copyright IBM Corp. 2008.                                         *
*                                                                   *
* US Government Users Restricted Rights - Use, duplication or       *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
*-------------------------------------------------------------------*
*
*-------------------------------------------------------------------*
*   START  OF WebSphere MQ  DCT ENTRIES
*-------------------------------------------------------------------*
*
*        MQER - System Log transient data queue
*
MQER     DFHDCT TYPE=INTRA,                                         *
               RSL=PUBLIC,                                         *
               DESTID=MQER,                                        *
               DESTFAC=FILE,                                       *
               TRANSID=MQER,                                       *
               TRIGLEV=1
*
*        MQXP - Message expiry transient data queue
*
MQXP     DFHDCT TYPE=INTRA,                                         *
               RSL=PUBLIC,                                         *
               DESTID=MQXP,                                        *
               DESTFAC=FILE,                                       *
               TRANSID=MQXP,                                       *
               TRIGLEV=1
*
*        MQIE - Instrumentation Events transient data queue
*
MQIE     DFHDCT TYPE=INTRA,                                         *
                RSL=PUBLIC,                                        *
                DESTID=MQIE,                                       *
                DESTFAC=FILE,                                      *
                TRANSID=MQIE,                                      *
                TRIGLEV=1
*
*        MQAC - Accounting Messages transient data queue.
*
```

```
MQAC     DFHDCT TYPE=INTRA,                                            *
               RSL=PUBLIC,                                             *
               DESTID=MQAC,                                            *
               DESTFAC=FILE,                                           *
               TRANSID=MQAC,                                           *
               TRIGLEV=1
*---------------------------------------------------------------------*
*   END  OF WebSphere MQ DCT ENTRIES
*---------------------------------------------------------------------*
```

Sample JCL file definition for CICS deck

```
*---------------------------------------------------------------------*
* Licensed Materials - Property of IBM                                *
* 5655-U97                                                            *
* Copyright IBM Corp. 2008                                            *
*                                                                     *
* US Government Users Restricted Rights - Use, duplication or         *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
*---------------------------------------------------------------------*
*   Sample JCL file definition for CICS deck                          *
*   The DLBL statements in this JCL correspond to entries in CICSFCT*
*   therefore if there are any new file ids to be added in here,      *
*   it must also be added into the corresponding JCL                  *
*                                                                     *
*   Fields marked with ?volid? and ?cat-name? must be changed to      *
*   suit customer own site specifications.                            *
*---------------------------------------------------------------------*
// DLBL MQFSSET,'WMQZVSE.MQFSSET',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFCNFG,'WMQZVSE.MQFCNFG',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFREOR,'WMQZVSE.MQFREOR',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFI001,'WMQZVSE.MQFI001',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFI002,'WMQZVSE.MQFI002',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFI003,'WMQZVSE.MQFI003',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFO001,'WMQZVSE.MQFO001',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFO002,'WMQZVSE.MQFO002',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFO003,'WMQZVSE.MQFO003',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFERR,'WMQZVSE.MQFERR',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFLOG,'WMQZVSE.MQFLOG',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFMON,'WMQZVSE.MQFMON',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFADMN,'WMQZVSE.MQFADMN',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFDEFS,'WMQZVSE.MQFDEFS',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFACCT,'WMQZVSE.MQFACCT',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
// DLBL MQFSTAT,'WMQZVSE.MQFSTAT',0,VSAM,CAT=?cat-name?
// EXTENT ,?volid?
/. c if you are using PCF then also customize following 2 labels
/. DLBL MQFACMD,'WMQZVSE.MQFACMD',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?
/. DLBL MQFARPY,'WMQZVSE.MQFARPY',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?
/. c if you require Instrumentation Events, customize following labels
/. DLBL MQFIEQE,'WMQZVSE.MQFIEQE',0,VSAM,CAT=?cat-name?
```

```
/. EXTENT ,?volid?
/. DLBL MQFIECE,'WMQZVSE.MQFIECE',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?
/. DLBL MQFIEPE,'WMQZVSE.MQFIEPE',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?
/. DLBL MQFIEME,'WMQZVSE.MQFIEME',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?
/. DLBL MQFIENE,'WMQZVSE.MQFIENE',0,VSAM,CAT=?cat-name?
/. EXTENT ,?volid?

*--------------------------------------------------------------------*
*   End of sample jcl file definition for cics deck                  *
*--------------------------------------------------------------------*
```

Sample JCL to create CICS CSD group

WebSphere MQ for z/VSE provides two sample jobs for the creation of CICS system definitions. These are:

```
MQJCSD.Z       -      For CICS for z/VSE
MQJCSD24.Z     -      For CICS Transaction Server
```

These JCL samples define WebSphere MQ programs, transactions and mapsets. In addition, for CICS Transaction Server, the MQJCSD24.Z sample provides CSD definitions for files. For CICS for z/VSE, files must be defined in the CICS File Control Table (FCT). JCL sample MQCICFCT.A is provided for this purpose.

These JCL samples must be tailored for your CICS environment. In particular, each program, transaction, mapset and file is defined to a specific group. This group is added to CICS group list. You must tailor the sample JCL to specify a group appropriate to your installation and specify your CICS startup group list.

Appendix B. Application Programming Reference

This appendix should be used in conjunction with the *WebSphere MQ Application Programming Reference* manual.

Structure data types

The following structure data types are supported by WebSphere MQ for z/VSE V3.0:

MQBMHO
Buffer to message handle options

MQCHARV
Variable-length string

MQCMHO
Create Message Handle Options

MQDH
Distribution header

MQDLH
Dead letter header

MQDMHO
Delete Message Handle Options

MQDMPO
Delete Message Property Options

MQGMO
Get message options

MQIMPO
Inquire Message Property Options

MQMD
Message descriptor

MQMDE
Message descriptor extension

MQMHBO
Message Handle To Buffer Options

MQOD
Object descriptor

MQOR
Object record

MQPD
Property Descriptor

MQPMO
Put message options

MQPMR
Put message record

MQRFH2
Rules and Formatting Header 2

MQRR
Response record

MQSMPO
Set Message Property Options

MQTM
Trigger message

MQXQH
Transmission-queue header

In addition, WebSphere MQ for z/VSE supports the MQCD and MQCXP in conjunction with channel exits. These data structures are described in "Channel-exit calls and data structures" on page 67. WebSphere MQ for z/VSE also supports data structures associated with MQ API Exits. These are described in Chapter 13, "API exits," on page 677.

The declarations of these structures are as described in the *WebSphere MQ Application Programming Reference*. This section describes how these data structures may vary in the z/VSE programming environment.

MQBMHO – Buffer to message handle options

The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. The structure is an input parameter on the MQBUFMH call.

Data in MQBMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields in the MQBMHO structure.

Table 32. Fields in MQBMHO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQBMHO

Here is a description of each field.

Options (MQLONG)
Options field. The value can be:

MQBMHO_DELETE_PROPERTIES
Properties that are added to the message handle are deleted from the buffer. If the call fails, no properties are deleted. Default options: If you do not need the option described, use the MQBMHO_NONE option.

MQBMHO_NONE
No options specified. This is always an input field.

The initial value of this field is MQBMHO_DELETE_PROPERTIES.

StrucId (MQLONG)
The structure identifier. The value must be:

MQBMHO_STRUC_ID
Identifier for buffer to message handle structure.

For the C programming language, the constant MQBMHO_STRUC_ID_ARRAY is also defined; this has the same value as MQBMHO_STRUC_ID but is an array of characters instead of a string.

This is always an input field.

The initial value of this field is MQBMHO_STRUC_ID.

Version (MQLONG)
The structure version number. The value must be:

MQBMHO_VERSION_1
Version number for buffer to message handle structure.

The following constant specifies the version number of the current version:

MQBMHO_CURRENT_VERSION
Current version of buffer to message handle structure.

This is always an input field.

The initial value of this field is MQBMHO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQBMHO.

Table 33. Initial values of fields in MQBMHO

Field name	Name of constant	Value of constant
StrucId	MQBMHO_STRUC_ID	'BMHO'
Version	MQBMHO_VERSION_1	1
Options	MQBMHO_NONE	0

Note: In the C programming language, the macro variable MQBMHO_DEFAULT contains the values shown in Table 33. To provide initial values for the fields in the structure, issue this statement:

```
MQBMHO MyBMHO = {MQBMHO_DEFAULT};
```

C declaration

```
typedef struct tagMQBMHO MQBMHO;
struct tagMQBMHO {
  MQCHAR4 StrucId; /* Structure identifier */
  MQLONG Version;  /* Structure version number */
  MQLONG Options;  /* Options that control the action of MQBUFMH */
};
```

COBOL declaration

```
** MQBMHO structure
 10  MQBMHO.
** Structure identifier
 15  MQBMHO-STRUCID                   PIC X(4).
** Structure version number
 15  MQBMHO-VERSION                   PIC S9(9) BINARY.
** Options that control the action of MQBUFMH
 15  MQBMHO-OPTIONS                   PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQBMHO based,
  3 StrucId          char(4)
    init(MQBMHO_STRUC_ID),          /* Structure identifier */
  3 Version          fixed bin(31)
    init(MQBMHO_VERSION_1),         /* Structure version number */
  3 Options          fixed bin(31)
    init(MQBMHO_DELETE_PROPERTIES); /* Options that control the */
                                    /* action of MQBUFMH */
```

MQCHARV – Variable-length string

Use the MQCHARV structure to describe a variable length string.

Data in the MQCHARV must be in the encoding of the local queue manager that is given by MQENC_NATIVE and the character set of the VSCCSID field within the structure. If the application is running as an MQ client, the structure must be in the encoding of the client. Some character sets have a representation that depends on the encoding. If VSCCSID is one of these character sets, the encoding used is the same encoding as that of the other fields in the MQCHARV.

The MQCHARV structure addresses data that might be discontiguous with the structure containing it. To address this data, fields declared with the pointer data type can be used. Be aware that COBOL does not support the pointer data type in all environments. Because of this, the data can also be addressed using fields that contain the offset of the data from the start of the structure containing the MQCHARV.

Fields

Here is a summary of the fields in the MQBMHO structure.

Table 34. Fields in MQCHARV

Field	Description
VSPtr	Pointer to the variable length string.
VSOffset	Offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure.
VSLength	The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.
VSBufSize	The size in bytes of the buffer addressed by the VSPtr or VSOffset field.
VSCCSID	The character set identifier of the variable-length string addressed by the VSPtr or VSOffset field.

Here is a discription of the fields.

VSBufSize (MQLONG)
The size in bytes of the buffer addressed by the VSPtr or VSOffset field.

When the MQCHARV structure is used as an output field on a function call, this field must be initialised with the length of the buffer provided. If the value of VSLength is greater than VSBufSize, then only VSBufSize

bytes of data are returned to the caller in the buffer. This value must be a value greater than or equal to zero, or the following special value applies:

MQVS_USE_VSLENGTH
When specified, the length of the buffer is taken from the VSLength field in the MQCHARV structure. Do not use this value when using the structure as an output field and a buffer is provided. This is the initial value of this field.

VSCCSID (MQLONG)
The character set identifier of the variable length-string addressed by the VSPtr or VSOffset field.

The initial value of this field is MQCCSI_APPL. This is defined by MQ to indicate that it should be changed by the queue manager to the true character set identifier of the queue manager, or the MQ client if running as an MQ client application. This is the same way as MQCCSI_Q_MGR behaves. As a result, the value MQCCSI_APPL is never associated with a variablelength string.

You can change the initial value of this field by defining a different value for the constant MQCCSI_APPL for your compile unit by the appropriate means for your application's programming language.

VSLength (MQLONG)
The length in bytes of the variable-length string addressed by the VSPtr or VSOffset field. The initial value of this field is 0. The value must be either greater than or equal to zero or the following special value applies:

MQVS_NULL_TERMINATED
If MQVS_NULL_TERMINATED is not specified, VSLength bytes are included as part of the string. If null characters are present, they do not delimit the string.

If MQVS_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null itself is not included as part of that string.

Note: The null character used to terminate a string (if MQVS_NULL_TERMINATED is specified) is a null from the codeset specified by VSCCSID.

For example, in UTF-16 (UCS-2 CCSIDs 1200 and 13488), this is the two-byteUnicode encoding where a null is represented by a 16-bit number of all zeros. In TF-16, it is common to find single bytes set to all zero which are part of characters (7-bit ASCII characters for instance), but the strings are only null-terminated when two "zero" bytes are found on an even-byte boundary. It is possible to get two "zero" bytes on an odd boundary when they are each part of valid characters. For example, X'01' X'00' X'00' X'30' are two valid Unicode characters and do not null-terminate the string.

VSOffset (MQLONG)
The offset in bytes of the variable-length string from the start of the MQCHARV, or the structure containing it. The offset can be positive or negative. You can use either the VSPtr or VSOffset field to specify the variable-length string, but not both.

When the MQCHARV structure is embedded within another structure, this value is the offset in bytes of the variable-length string from the start of the structure that contains this MQCHARV structure.

When the MQCHARV structure is not embedded within another structure (for example, if it is specified as a parameter on a function call), the offset is relative to the start of the MQCHARV structure.

The initial value of this field is 0.

VSPtr (MQPTR)
A pointer to the variable length string.

You can use either the VSPtr or VSOffset field to specify the variable-length string, but not both.

The initial value of this field is a null pointer or null bytes.

Initial values and language declarations

Here are the initial values and language declarations for MQBCHARV.

Table 35. Initial values of fields in MQBCHARV

Field name	Name of constant	Value of constant
VSPtr	None	Null pointer or null bytes
VSOffset	None	0
VSBufSize	MQVS_USE_VSLENGTH	-1
VSLength	None	0
VSCCSID	MQCCSI_APPL	-3

Note: In the C programming language, the macro variable MQCHARV_DEFAULT contains the values listed above. To provide initial values for the fields in the structure, issue this statement:

```
MQCHARV MyVarStr = {MQCHARV_DEFAULT};
```

C declaration

```
struct tagMQCHARV {
  MQPTR             VSPtr;         /* addr of variable length string */
  MQLONG            VSOffset;      /* offset of variable string      */
  MQLONG            VSBufSize;     /* Size of buffer                 */
  MQLONG            VSLength;      /* length of variable string      */
  MQLONG            VSCCSID;       /* CCSID of variable string       */
};
```

COBOL declaration

```
** MQCHARV structure
 10  MQCHARV.
** Address of variable length string
 15  MQCHARV-VSPTR                    USAGE POINTER.
** Offset of variable length string
 15  MQCHARV-VSOFFSET                 PIC S9(9) BINARY.
** Size of buffer
 15  MQCHARV-VSBUFSIZE                PIC S9(9) BINARY.
** Length of variable length string
 15  MQCHARV-VSLENGTH                 PIC S9(9) BINARY.
** CCSID of variable length string
 15  MQCHARV-VSCCSID                  PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQCHARV based,
  3 VSPtr           pointer
    init(null()),           /* Address of variable length string */
  3 VSOffset        fixed bin(31)
    init(0),                /* Offset of variable length string */
  3 VSBufSize       fixed bin(31)
    init(0),                /* Size of buffer */
  3 VSLength        fixed bin(31)
    init(0),                /* Length of variable length string */
  3 VSCCSID         fixed bin(31)
    init(MQCCSI_APPL);      /* CCSID of variable length string */
```

MQCMHO – Create message handle options

The MQCMHO structure allows applications to specify options that control how message handles are created. The structure is an input parameter on the MQCRTMH call.

Data in MQCMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields in the MQCMHO structure.

Table 36. Fields in MQCMHO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQCMHO

Here is a description of the fields.

Options (MQLONG)
One of these options can be specified:

MQCMHO_VALIDATE
When MQSETMP is called to set a property in this message handle, the property name is validated to ensure that it:

- Contains no invalid characters.
- Does not begin with "JMS" or "usr.JMS" except for these:
  - JMSCorrelationID
  - JMSReplyTo
  - JMSType
  - JMSXGroupID
  - JMSXGroupSeq

  These names are reserved for JMS properties.
- Is not one of these keywords, in any mixture of upper or lowercase:
  AND
  BETWEEN
  ESCAPE
  FALSE

IN
IS
LIKE
NOT
NULL
OR
TRUE

- Does not begin with "Body." or "Root." (except for "Root.MQMD.").

If the property is MQ-defined (mq.*) and the name is recognized, the property descriptor fields are set to the correct values for the property. If the property is not recognized, the Support field of the property descriptor is set to MQPD_OPTIONAL.

MQCMHO_DEFAULT_VALIDATION
Specifies that the default level of validation of property names should occur.

The default level of validation is equivalent to that specified by MQCMHO_VALIDATE.

In a future release, an administrative option may be defined which will change the level of validation that occurs when MQCMHO_DEFAULT_VALIDATION is defined. This is the default value.

MQCMHO_NO_VALIDATION
No validation on the property name occurs.

See the description of MQCMHO_VALIDATE.

Default option: If none of the options described above is required, the following option can be used:

MQCMHO_NONE
All options assume their default values. Use this value to indicate that no other options have been specified. MQCMHO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is MQCMHO_DEFAULT_VALIDATION.

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQCMHO_STRUC_ID
Identifier for create message handle options structure.

For the C programming language, the constant MQCMHO_STRUC_ID_ARRAY is also defined; this has the same value as MQCMHO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCMHO_STRUC_ID.

Version (MQLONG)
This is the structure version number; the value must be:

MQCMHO_VERSION_1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

MQCMHO_CURRENT_VERSION
Current version of create message handle options structure.

This is always an input field. The initial value of this field is MQCMHO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQCMHO.

Table 37. Initial values of fields in MQBCCMHO

Field name	Name of constant	Value of constant
StrucId	MQCMHO_STRUC_ID	'CMHO'
Version	MQCMHO_VERSION_1	1
Options	MQCMHO_DEFAULT_VALIDATION	0

Note: In the C programming language, the macro variable MQCMHO_DEFAULT contains the values listed above. Issue this statement to provide initial values for the fields in the structure:

```
MQCMHO MyCMHO = {MQCMHO_DEFAULT};
```

C declaration

```
struct tagMQCMHO {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQCRTMH */
};
```

COBOL declaration

```
** MQCMHO structure
 10  MQCMHO.
** Structure identifier
 15  MQCMHO-STRUCID                    PIC X(4).
** Structure version number
 15  MQCMHO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQCRTMH
 15  MQCMHO-OPTIONS                    PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQCMHO based,
  3 StrucId          char(4)
    init(MQCMHO_STRUC_ID),          /* Structure identifier */
  3 Version          fixed bin(31)
    init(MQCMHO_VERSION_1),         /* Structure version number */
  3 Options          fixed bin(31)
    init(MQCMHO_DEFAULT_VALIDATION); /* Options that control the */
                                    /* action of MQCRTMH */
```

MQDLH – Dead-letter header

The MQDLH structure describes the information that prefixes the application message data of messages on the dead-letter (undelivered-message) queue. A message can arrive on the dead-letter queue either because the queue manager or message channel agent has redirected it to the queue, or because an application has put the message directly on the queue.

Applications that put messages directly on the dead-letter queue must prefix the message data with an MQDLH structure, and initialize the fields with appropriate values. However, the queue manager does not require that an MQDLH structure be present, or that valid values have been specified for the fields.

If a message is too long to put on the dead-letter queue, the application must do one of the following:

- Truncate the message data to fit on the dead-letter queue.
- Record the message on auxiliary storage and place an exception report message on the dead-letter queue indicating this.
- Discard the message and return an error to its originator. If the message is (or might be) a critical message, do this only if it is known that the originator still has a copy of the message; for example, a message received by a message channel agent from a communication channel.

Which of the above is appropriate (if any) depends on the design of the application.

The queue manager performs special processing when a message that is a segment is put with an MQDLH structure at the front; see the description of the MQMDE structure for further details.

Fields

Here is a summary of the fields.

Table 38. Fields in MQDLH

Field	Description
StrucId	Structure identifier
Version	Structure version number
Reason	Reason message arrived on dead-letter queue
DestQName	Name of original destination queue
DestQMgrName	Name of original destination queue manager
Encoding	Numeric encoding of data that follows MQDLH
CodedCharSetId	Character set identifier of data that follows MQDLH
Format	Format name of data that follows MQDLH
PutApplType	Type of application that put message on dead-letter queue

Table 38. Fields in MQDLH (continued)

Field	Description
PutApplName	Name of application that put message on dead-letter queue
PutDate	Date when message was put on dead-letter queue
PutTime	Time when message was put on dead-letter queue

Here is a description of the fields.

CodedCharSetId (MQLONG)
This is the character set identifier of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to character data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQCCSI_UNDEFINED.

DestQMgrName (MQCHAR48)
This is the name of the queue manager that was the original destination for the message.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

DestQName (MQCHAR48)
This is the name of the message queue that was the original destination for the message.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Encoding (MQLONG)
This is the numeric encoding of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to numeric data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Format (MQCHAR8)
This is the format name of the data that follows the MQDLH structure (usually the data from the original message).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the Format field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

PutApplName (MQCHAR28)
This is the name of the application that put the message on the dead-letter (undelivered-message) queue.

The format of the name depends on the PutApplType field. See also the description of the PutApplName field in "MQMD – Message descriptor" on page 774.

If the queue manager redirects the message to the dead-letter queue, PutApplName contains the first 28 characters of the queue-manager name, padded with blanks if necessary.

The length of this field is given by MQ_PUT_APPL_NAME_LENGTH. The initial value of this field is the null string in C, and 28 blank characters in other programming languages.

PutApplType (MQLONG)
This is the type of application that put the message on the dead- letter (undelivered-message) queue. This field has the same meaning as the PutApplType field in the message descriptor MQMD (see "MQMD – Message descriptor" on page 774 for details).

If the queue manager redirects the message to the dead-letter queue, PutApplType has the value MQAT_QMGR.

The initial value of this field is 0.

PutDate (MQCHAR8)
The date when the message was put on the dead-letter (undelivered-message) queue. The format used for the date when this field is generated by the queue manager is YYYYMMDD, where the characters represent:

YYYY Year (four numeric digits)

MM Month of year (01 through 12)

DD Day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the PutDate and PutTime fields, subject to the system clock being set accurately to GMT.

The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8)
This is time when the message was put on the dead-letter (undelivered-message) queue. The format used for the time when this field is generated by the queue manager is HHMMSSTH, where the characters represent:

HH Hours (00 through 23)

MM Minutes (00 through 59)

SS Seconds (00 through 59)

T Tenths of a second (0 through 9)

H Hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in PutTime. This happens when leap seconds are inserted into the global time standard.
Greenwich Mean Time (GMT) is used for the PutDate and PutTime fields,

subject to the system clock being set accurately to GMT. The length of this field is given by MQ_PUT_TIME_LENGTH.

The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

Reason (MQLONG)
This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It should be one of the MQFB_* or MQRC_* values (for example, MQRC_Q_FULL). See the description of the Feedback field in "MQMD – Message descriptor" on page 774 for details of the common MQFB_* values that can occur.

The initial value of this field is MQRC_NONE.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQDLH_STRUC_ID
Identifier for dead-letter header structure

For the C programming language, the constant MQDLH_STRUC_ID_ARRAY is also defined; this has the same value as MQDLH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDLH_STRUC_ID.

Version (MQLONG)
This is the structure version number. The value must be:

MQDLH_VERSION_1
Version number for dead-letter header structure

The following constant specifies the version number of the current version:

MQDLH_CURRENT_VERSION
Current version of dead-letter header structure

The initial value of this field is MQDLH_VERSION_1.

C declaration

```
typedef struct tagMQDLH MQDLH;
struct tagMQDLH
{
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Reason;          /* Reason message arrived on dead-letter queue */
  MQCHAR48  DestQName;       /* Name of original destination queue */
  MQCHAR48  DestQMgrName;    /* Name of original destination queue manager */
  MQLONG    Encoding;        /* Numeric encoding of data that follows MQDLH */
  MQLONG    CodedCharSetId;  /* Character set id of data that follows MQDLH */
  MQCHAR8   Format;          /* Format name of data that follows MQDLH */
  MQLONG    PutApplType;     /* Type of application that put message on dlq */
  MQCHAR28  PutApplName;     /* Name of application that put message on dlq */
  MQCHAR8   PutDate;         /* Date when message was put on dlq */
  MQCHAR8   PutTime;         /* Time when message was put on the dlq */
};
```

COBOL declaration

```
** MQDLH structure
  10 MQDLH.
** Structure identifier
   15 MQDLH-STRUCID PIC X(4).
** Structure version number
   15 MQDLH-VERSION PIC S9(9) BINARY.
** Reason message arrived on dead-letter queue
   15 MQDLH-REASON PIC S9(9) BINARY.
** Name of original destination queue
   15 MQDLH-DESTQNAME PIC X(48).
** Name of original destination queue manager
   15 MQDLH-DESTQMGRNAME PIC X(48).
** Numeric encoding of data that follows MQDLH
   15 MQDLH-ENCODING PIC S9(9) BINARY.
** Character set identifier of data that follows MQDLH
   15 MQDLH-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows MQDLH
   15 MQDLH-FORMAT PIC X(8).
** Type of application that put message on dlq
   15 MQDLH-PUTAPPLTYPE PIC S9(9) BINARY.
** Name of application that put message on dlq
   15 MQDLH-PUTAPPLNAME PIC X(28).
** Date when message was put on dlq
   15 MQDLH-PUTDATE PIC X(8).
** Time when message was put on the dlq
   15 MQDLH-PUTTIME PIC X(8).
```

PL/I declaration

```
dcl
 1 MQDLH based,
  3 StrucId char(4),               /* Structure identifier */
  3 Version fixed bin(31),         /* Structure version number */
  3 Reason fixed bin(31),          /* Reason message arrived on dead-letter queue */
  3 DestQName char(48),            /* Name of original destination queue */
  3 DestQMgrName char(48),         /* Name of original destination queue manager */
  3 Encoding fixed bin(31),        /* Numeric encoding of data that follows MQDLH */
  3 CodedCharSetId fixed bin(31),  /* Character set id of data that follows MQDLH */
  3 Format char(8),                /* Format name of data that follows MQDLH */
  3 PutApplType fixed bin(31),     /* Type of application that put message on dlq */
  3 PutApplName char(28),          /* Name of application that put message on dlq */
  3 PutDate char(8),               /* Date when message was put on dlq */
  3 PutTime char(8);               /* Time when message was put on the dlq */
```

MQDH – Distribution Header

The MQDH structure describes the additional data that is present in a message, when that message is a distribution-list message stored on a transmission queue. A distribution-list message is a message that is sent to multiple destination queues. The additional data consists of the MQDH structure, followed by an array of MQOR records and an array of MQPMR records.

This structure is for use by specialized applications that put messages directly on transmission queues, or which remove messages from transmission queues (for example: message channel agents).

This structure should not be used by normal applications which simply want to put messages to distribution lists. Those applications should use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Format name: MQFMT_DIST_HEADER

Character set and encoding: Character data in MQDH must be in the character set of the local queue manager; this is given by the CodedCharSetId queue-manager attribute. Numeric data in MQDH must be in the native machine encoding; this is given by the value of MQENC_NATIVE for the C programming language.

The character set and encoding of the MQDH must be set into the CodedCharSetId and Encoding fields in:

- The MQMD (if the MQDH structure is at the start of the message data), or
- The header structure that precedes the MQDH structure (all other cases).

When an application puts a message to a distribution list, and some or all of the destinations are remote, the queue manager prefixes the application message data with the MQXQH and MQDH structures, and places the message on the relevant transmission queue. The data therefore occurs in the following sequence when the message is on a transmission queue:

- MQXQH structure.
- MQDH structure plus arrays of MQOR and MQPMR records.
- Application message data.

Depending on the destinations, more than one such message may be generated by the queue manager, and placed on different transmission queues. In this case, the MQDH structures in those messages identify different subsets of the destinations defined by the distribution list opened by the application.

An application that puts a distribution-list message directly on a transmission queue must conform to the sequence described above, and must ensure that the MQDH structure is correct. If the MQDH structure is not valid, the queue manager may choose to fail the MQPUT or MQPUT1 call with reason code MQRC_DH_ERROR.

Messages can be stored on a queue in distribution-list form only if the queue is defined as being able to support distribution list messages. If an application puts a distribution-list message directly on a queue that does not support distribution lists, the queue manager splits the distribution list message into individual messages, and places those on the queue instead.

Fields

Here is a summary of the fields.

Table 39. Fields in MQDH

Field	Description
StrucId	Structure identifier
Version	Structure version number
StrucLength	Length of MQDH structure plus following records
Encoding	Encoding of message data
CodedCharSetId	Coded character-set identifier of message data
Format	Format name of message data
Flags	General flags
PutMsgRecFields	Flags indicating which MQPMR fields are present
RecsPresent	Number of object records present
ObjectRecOffset	Offset of first object record from start of MQDH

Table 39. Fields in MQDH (continued)

Field	Description
PutMsgRecOffset	Offset of first put message record from start of MQDH

Here is a description of the fields.

CodedCharSetId (MQLONG)
Character set identifier of data that follows the array of MQPMR records.

This specifies the character set identifier of the data that follows the arrays of MQOR and MQPMR records; it does not apply to character data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Inherit character-set identifier of this structure. Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)
Numeric encoding of the data that follows the array of MQPMR records.

This specifies the numeric encoding of the data that follows the arrays of MQOR and MQPMR records; it does not apply to numeric data in the MQDH structure itself

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG)
General flags.

The following flag can be specified:

- MQDHF_NEW_MSG_IDS

  Generate new message identifiers. This flag indicates that a new message identifier is to be generated for each destination in the distribution list. This can be set only when there are no put-message records present, or when the records are present, but they do not contain the MsgId field.

Using this flag defers generation of the message identifiers until the last possible moment, namely the moment when the distribution-list message is finally split into individual messages. This minimizes the amount of control information that must flow with the distribution-list message.

When an application puts a message to a distribution list, the queue manager sets MQDHF_NEW_MSG_IDS in the MQDH it generates when both of the following are true:

- There are no put-message records provided by the application, or the records provided do not contain the MsgId field.

- The MsgId field in MQMD is MQMI_NONE, or the Options field in MQPMO includes MQPMO_NEW_MSG_ID.

If no flags are needed, the following can be specified:

MQDHF_NONE
No flags. This constant indicates that no flags have been specified. MQDHF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQDHF_NONE.

Format (MQCHAR8)
Format name of the data that follows the array of MQPMR records.

This specifies the format name of the data that follows the arrays of MQOD and MQPMR records, or whichever occurs last.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the Format field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectRecOffset (MQLONG)
Offset of first object record from the start of MQDH.

This field gives the offset in bytes of the first record in the array of MQOR object records containing the names of the destination queues. There are RecsPresent records in this array. These records, plus any bytes skipped between the first object record and the previous field, are included in the length given by the StrucLength field.

A distribution list must always contain at least one destination, so ObjectRecOffset must always be greater than zero.

The initial value of this field is 0.

PutMsgRecFields (MQLONG)
Flags indicating which MQPMR are present.

Zero or more of the following flags can be specified:

- MQPMRF_MSG_ID Message-identifier field is present.
- MQPMRF_CORREL_ID Correlation-identifier field is present.
- MQPMRF_GROUP_ID Group-identifier field is present.
- MQPMRF_FEEDBACK Feedback field is present.
- MQPMRF_ACCOUNTING_TOKEN Accounting-token field is present.

If no MQPMR fields are present, the following can be specified:

MQPMRF_NONE
No put-message record fields are present. MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQPMRDF_NONE.

PutMsgRecOffset (MQLONG)
Offset of first put message record from start of MQDH.

This field gives the offset in bytes of the first record in the array of MQPMR put message records containing the message properties. If

present, there are RecsPresent records in this array. These records, plus any bytes skipped between the first put message record and the previous field, are included in the length given by the StrucLength field.

Put message records are optional; if no records are provided, PutMsgrecOffset is zero, and PutMsgRecFields has the value MQPMRF_NONE.

The initial value of this field is O.

RecsPresent (MQLONG)
Number of object records present.

This defines the number of destinations. A distribution list must always contain at least one destination, so RecsPresent must always be greater than zero.

The initial value of this field is O.

StrucId (MQCHAR4)
Structure identifier.

The value must be:

MQDG_STRUC_ID
Identifier for distribution header structure. For the C programming language, the constant MQDH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDH_STRUC_ID.

StrucLength (MQLONG)
Length of MQDH structure plus following records.

This is the number of bytes from the start of the MQDH structure to the start of the message data following the arrays of MQOR and MQPMR records. The data occurs in the following sequence:

- MQDH_structure.
- Array of MQOR records.
- Array of MQPMR records.
- Message data.

The arrays of MQOR and MQPMR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or more of the MQDH structure, the arrays of records, and the message data, these unused bytes must be included in the value of StrucLength. The content of these bytes, however, is not preserved by the queue manager. It is valid for the array of MQPMR records to precede the array of MQPMR records.

The initial value of this field is 0.

Version (MQLONG)
This is the structure version number; the value must be one of the following:

MQDH_VERSION_1
Version-1 object descriptor structure.

The following constant specifies the version number of the current version:

MQDH_CURRENT_VERSION
Current version of object descriptor structure. For WebSphere MQ for z/VSE, this field defaults to MQOD_VERSION_1.

The initial value of this field is MQDH_VERSION_1.

C declaration

```
struct tagMQDH
{
  MQCHAR4   StrucId;
  MQLONG    Version;
  MQLONG    StrucLength;
  MQLONG    Encoding;
  MQLONG    CodedCharSetId;
  MQCHAR8   Format;
  MQLONG    Flags;
  MQLONG    PutMsgRecFields;
  MQLONG    RecsPresent;
  MQLONG    ObjectRecOffset;
  MQLONG    PutMsgRecOffset;

};
typedef struct tagMQDH MQDH;
```

COBOL declaration

```
  ** MQDH structure
     10 MQDH.
  ** Structure identifier
     15 MQDH-STRUCID PIC X(4).VALUE 'DH'.
  ** Structure version number
     15 MQDH-VERSION PIC S9(9) BINARY VALUE 1.
  ** Length of MQDH structure plus following MQOR and MQPMR records
     15 MQDH-STRUCLENGTH PIC S9(9) BINARY VALUE 0.
  ** Numeric encoding of data that follows the MQOR and MQPMR
  ** records
     15 MQDH-ENDCODING PIC S9(9) BINARY VALUE 0.
  ** Character set identifier of data that follows the MQOR and
  ** MQPMR records
     15 MQDH-CODEDCHARSETID PIC S9(9) BINARY VALUE 0.
  ** Format name of data that follows the MQOR and MQPMR records
     15 MQDH-FORMAT PIC X(8) VALUE SPACES.
  ** General flags
     15 MQDH-FLAGS PIC S9(9) BINARY VALUE O.
  ** Flags indicating which MQPMR fields are present
     15 MQDH-PUTMSGRECFIELDS PIC S9(9) BINARY VALUE 0.
  ** Number of MQOR records present
     15 MQDH-RECSPRESENT PIC S9(9)BINARY VALUE 0.
  ** Offset of first MQOR record from start of MQDH
     15 MQDH-OBJECTRECOFFSET PIC S9(9) BINARY VALUE 0.
  ** Offset of first MQPMR record from start of MQDH
     15 MQDH-PUTMSGRECOFFSET PIC S9(9) BINARY VALUE 0.
```

PL/I declaration

```
dcl
 1 MQDH based,
  3 StrucId           char(4)
      init(MQDH_STRUC_ID),          /* Structure identifier */
  3 Version           fixed bin(31)
      init(MQDH_VERSION_1),         /* Structure version number */
  3 StrucLength       fixed bin(31)
      init(0),                      /* Length of MQDH structure plus
                                       following MQOR and MQPMR
                                       records */

  3 Encoding          fixed bin(31)
      init(0),                      /* Numeric encoding of data that
                                       follows the MQOR and MQPMR
                                       records */

  3 CodedCharSetId    fixed bin(31)
      init(MQCCSI_UNDEFINED),       /* Character set identifier of data
                                       that follows the MQOR and MQPMR
                                       records */

  3 Format            char(8)
      init(MQFMT_NONE),             /* Format name of data that follows
                                       the MQOR and MQPMR records */

  3 Flags             fixed bin(31)
      init(MQDHF_NONE)              /* General flags */
  3 PutMsgRecFields   fixed bin(31)
      init(MQPMRF_NONE),            /* Flags indicating which MQPMR
                                       fields are present */

  3 RecsPresent       fixed bin(31)
      init(0),                      /* Number of MQOR records
                                       present */

  3 ObjectRecOffset   fixed bin(31)
      init(0),                      /* Offset of first MQOR record from
                                       start of MQDH */

  3 PutMsgRecOffset   fixed bin(31)
      init(0),                      /* Offset of first MQPMR record
                                       from start of MQDH */
```

MQDMHO – Delete message handle options

The MQDMHO structure allows applications to specify options that control how message handles are deleted. The structure is an input parameter on the MQDLTMH call.

Data in MQDMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 40. Fields in MQDMHO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQDMHO

Here is a description of the fields.

Options (MQLONG)
The value must be:

MQDMHO_NONE
No options specified.

This is always an input field. The initial value of this field is MQDMHO_NONE.

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQDMHO_STRUC_ID
Identifier for create message handle options structure.

For the C programming language, the constant MQDMHO_STRUC_ID_ARRAY is also defined; this has the same value as MQDMHO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQDMHO_STRUC_ID.

Version (MQLONG)
This is the structure version number; the value must be:

MQDMHO_VERSION_1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

MQDMHO_CURRENT_VERSION
Current version of create message handle options structure.

This is always an input field. The initial value of this field is MQDMHO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQDMHO.

Table 41. Initial values of fields in MQDMHO

Field name	Name of constant	Value of constant
StrucId	MQDMHO_STRUC_ID	'DMHO'
Version	MQDMHO_VERSION_1	1
Options	MQDMHO_NONE	0

Note: In the C programming language, the macro variable MQDMHO_DEFAULT contains the values shown in Table 33 on page 719. To provide initial values for the fields in the structure, issue this statement:

```
MQDMHO MyDMHO = {MQDMHO_DEFAULT};
```

C declaration

```
struct tagMQDMHO {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQDLTMH */
};
```

COBOL declaration

```
** MQDMHO structure
 10  MQDMHO.
** Structure identifier
 15  MQDMHO-STRUCID                    PIC X(4).
** Structure version number
 15  MQDMHO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQDLTMH
 15  MQDMHO-OPTIONS                    PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQDMHO based,
  3 StrucId        char(4)
    init(MQDMHO_STRUC_ID),  /* Structure identifier */
  3 Version        fixed bin(31)
    init(MQDMHO_VERSION_1), /* Structure version number */
  3 Options        fixed bin(31)
    init(MQDMHO_NONE);      /* Options that control the action of */
                            /* MQDLTMH */
```

MQDMPO – Delete message properties options

The MQDMPO structure allows applications to specify options that control how properties of messages are deleted. The structure is an input parameter on the MQDLTMP call.

Data in MQDMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 42. Fields in MQDMPO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQDMPO

Here is a description of the fields.

Options (MQLONG)
Options field. Delete message property options structure.

Location options: The following options relate to the relative location of the property compared to the property cursor.

MQDMPO_DEL_FIRST
Deletes the first property that matches the specified name.

MQDMPO_DEL_NEXT
Deletes the next property that matches the specified name, continuing the search from the property cursor. If this is the first MQDLTMP call for the specified name, the first property that matches the specified name is deleted.

If the property under the cursor has been deleted, MQINQMP deletes the next matching property following the one that has been deleted.

If a property is added that matches the specified name while iteration is in progress, the property might be deleted during the completion of the iteration. The property is deleted once the iteration is restarted with MQDMPO_DEL_FIRST.

MQDMPO_DEL_PROP_UNDER_CURSOR
Deletes the property pointed to by the property cursor. That is, the property that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the MsgHandle field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established, or if the property pointer to by the property cursor has already been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

If none of the options described above is required, the following option can be used:

MQDMPO_NONE
No options specified.

This is always an input field. The initial value of this field is MQDMPO_DEL_FIRST.

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQDMPO_STRUC_ID
Identifier for create message handle options structure.

For the C programming language, the constant MQDMPO_STRUC_ID_ARRAY is also defined. This has the same value as MQDMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQDMPO_STRUC_ID.

Version (MQLONG)
This is the structure version number. The value must be:

MQDMPO_VERSION_1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

MQDMPO_CURRENT_VERSION
Current version of create message handle options structure.

This is always an input field. The initial value of this field is MQDMPO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQDMPO.

Table 43. Initial values of fields in MQDMPO

Field name	Name of constant	Value of constant
StrucId	MQDMPO_STRUC_ID	'DMPO'
Version	MQDMPO_VERSION_1	1
Options	MQDMPO_NONE	0

Note: In the C programming language, the macro variable MQDMPO_DEFAULT contains the values shown in Table 43. To provide initial values for the fields in the structure, issue this statement:

```
MQDMPO MyDMPO = {MQDMPO_DEFAULT};
```

C declaration

```
struct tagMQDMPO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQDLTMP */
};
```

COBOL declaration

```
** MQDMPO structure
 10  MQDMPO.
** Structure identifier
 15  MQDMPO-STRUCID                    PIC X(4).
** Structure version number
 15  MQDMPO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQDLTMP
 15  MQDMPO-OPTIONS                    PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQDMPO based,
  3 StrucId        char(4)
    init(MQDMPO_STRUC_ID),  /* Structure identifier */
  3 Version        fixed bin(31)
    init(MQDMPO_VERSION_1), /* Structure version number */
  3 Options        fixed bin(31)
    init(MQDMPO_DEL_FIRST); /* Options that control the action of */
                            /* MQDLTMP */
```

MQGMO – Get message options

The MQGMO structure allows the application to specify options that control how messages are removed from queues. The structure is an input/output parameter on the MQGET call.

Fields

Here is a summary of the fields.

Table 44. Fields in MQGMO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options that control the action of MQGET
WaitInterval	Wait interval
Signal1	Signal
Signal2	Signal identifier
ResolvedQName	Resolved name of destination queue
Note: The remaining fields are ignored if Version is less than MQGMO_VERSION_2.	
MatchOptions	Options controlling selection criteria used for MQGET
GroupStatus	Flag indicating whether message retrieved is in a group
SegmentStatus	Flag indicating whether message retrieved is a segment of a logical message
Segmentation	Flag indicating whether further segmentation is allowed for the message retrieved
Reserved1	Reserved
Note: The remaining fields are ignored if Version is less than MQGMO_VERSION_3.	
MsgToken	Message token
ReturnedLength	Length of message data returned (bytes)
Note: The remaining fields are ignored if Version is less than MQGMO_VERSION_4.	
Reserved2	Reserved
MsgHandle	The handle to a message that is to be populated with the properties of the message being retrieved from the queue.

The MQGMO structure contains the following fields; the fields are described in alphabetic order:

GroupStatus (MQCHAR)
This flag indicates whether the message retrieved is in a group. It has one of the following values:

MQGS_NOT_IN_GROUP
Message is not in a group.

MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQGS_LAST_MSG_IN_GROUP
Message is the last in the group. This is also the value returned if the group consists of only one message.

This is an output field. The initial value of this field is MQGS_NOT_IN_GROUP. This field is ignored if Version is less than MQGMO_VERSION_2.

MatchOptions (MQLONG)
These options allow the application to choose which fields in the MsgDesc parameter to use to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the MsgDesc parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that MsgDesc parameter on the MQGET call. Fields for which the corresponding match option is not specified are ignored when selecting the message to be returned. If you specify no selection criteria on the MQGET call (that is, any message is acceptable), set MatchOptions to MQMO_NONE.

If you specify MQGMO_LOGICAL_ORDER, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have MsgSeqNumber equal to 1 and Offset equal to 0 are eligible for return. In this situation, you can use one or more of the following match options to select which of the eligible messages is returned:
  MQMO_MATCH_MSG_ID
  MQMO_MATCH_CORREL_ID
  MQMO_MATCH_GROUP_ID
- If there is a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying MQMO_* options.

In both of the above cases, you can specify match options that do not apply, but the value of the relevant field in the MsgDesc parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code MQRC_MATCH_OPTIONS_ERROR is this condition is not satisfied.

MatchOptions is ignored if you specify either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR. You can specify one or more of the following match options:

MQMO_MATCH_MSG_ID
The message to be retrieved must have a message identifier that matches the value of the MsgId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the MsgId field in the MsgDesc parameter is ignored, and any message identifier will match.

The message identifier MQMI_NONE is a special value that matches any message identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_MSG_ID with MQMI_NONE is the same as not specifying MQMO_MATCH_MSG_ID.

MQMO_MATCH_CORREL_ID
The message to be retrieved must have a correlation identifier that matches the value of the CorrelId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If you omit this option, the CorrelId field in the MsgDesc parameter is ignored, and any correlation identifier will match.

The correlation identifier MQCI_NONE is a special value that matches any correlation identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_CORREL_ID with MQCI_NONE is the same as not specifying MQMO_MATCH_CORREL_ID.

MQMO_MATCH_GROUP_ID
The message to be retrieved must have a group identifier that matches the value of the GroupId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the GroupId field in the MsgDesc parameter is ignored, and any group identifier will match.

The group identifier MQGI_NONE is a special value that matches any group identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_GROUP_ID with MQGI_NONE is the same as not specifying MQMO_MATCH_GROUP_ID.

MQMO_MATCH_MSG_SEQ_NUMBER
The message to be retrieved must have a message sequence number that matches the value of the MsgSeqNumber field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If you omit this option, the MsgSeqNumber field in the MsgDesc parameter is ignored, and any message sequence number will match.

MQMO_MATCH_OFFSET
The message to be retrieved must have an offset that matches the value of the Offset field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If you omit this option is not specified, the Offset field in the MsgDesc parameter is ignored, and any offset will match.

MQMO_NONE
Use no matches in selecting the message to be returned; all messages on the queue are eligible for retrieval (but subject to control by the MQGMO_ALL_MSGS_AVAILABLE, MQGMO_ALL_SEGMENTS_AVAILABLE, and MQGMO_COMPLETE_MSG options).

MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of this field is MQMO_MATCH_MSG_ID with MQMO_MATCH_CORREL_ID. This field is ignored if Version is less than MQGMO_VERSION_2.

MsgHandle (MQHMSG)
If the MQGMO_PROPERTIES_AS_Q_DEF option is specified and the PropertyControl queue attribute is not set to MQPROP_FORCE_MQRFH2, then this is the handle to a message which will be populated with the properties of the message being retrieved from the queue. The handle is created by an MQCRTMH call. Any properties already associated with the handle will be cleared before retrieving a message.

The following value can also be specified:

MQHM_NONE
No message handle supplied.

No message descriptor is required on the MQGET call if a valid message handle is supplied and used on output to contain the message properties. The message descriptor associated with the message handle is used for input fields.

If a message descriptor is specified on the MQGET call, it always takes precedence over the message descriptor associated with a message handle.

If MQGMO_PROPERTIES_FORCE_MQRFH2 is specified, or the MQGMO_PROPERTIES_AS_Q_DEF is specified and the PropertyControl queue attribute is MQPROP_FORCE_MQRFH2, then the call fails with reason code MQRC_MD_ERROR when no message descriptor parameter is specified.

On return from the MQGET call, the properties and message descriptor associated with this message handle are updated to reflect the state of the message retrieved (as well as the message descriptor if one was supplied on the MQGET call). The properties of the message can then be inquired using the MQINQMP call.

Except for message descriptor extensions, when present, a property that can be inquired with the MQINQMP call is not contained in the message data; if the message on the queue contained properties in the message data, these are removed from the message data before the data is returned to the application.

If no message handle is provided or Version is less than MQGMO_VERSION_4, then you must supply a valid message descriptor on the MQGET call. Any message properties (except those contained in the message descriptor) are returned in the message data subject to the value of the property options in the MQGMO structure and the PropertyControl queue attribute.

This is an always an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQGMO_VERSION_4.

Note: MQHM_NONE is defined as an 8-byte character field in C and PL/I, and defined in COBOL as PIC S9(18) COMP.

MsgToken (MQBYTE16)
Reserved field in WebSphere MQ for z/VSE.

Options (MQLONG)

These options control the action of MQGET. You can specify none or more of the options described below. If you need more than one the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

The following *wait options* relate to waiting for messages to arrive on the queue:

MQGMO_WAIT

The application waits until a suitable message arrives. The maximum time that the application waits is specified in WaitInterval.

If MQGET requests are inhibited, the wait is canceled and the call completes with MQCC_FAILED and reason code MQRC_GET_INHIBITED, regardless of whether there are suitable messages on the queue.

You can use this option with the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT options.

If several applications are waiting on the same shared queue, the applications that are activated when a suitable message arrives are described below.

Note: In the description below, a browse MQGET call is one that specifies one of the browse options, but not MQGMO_LOCK; an MQGET call specifying the MQGMO_LOCK option is treated as a nonbrowse call.

- If one or more nonbrowse MQGET calls is waiting, but no browse MQGET calls are waiting, one is activated.
- If one or more browse MQGET calls is waiting, but no nonbrowse MQGET calls are waiting, all are activated.
- If one or more nonbrowse MQGET calls, and one or more browse MQGET calls are waiting, one nonbrowse MQGET call is activated, and none, some, or all of the browse MQGET calls. (The number of browse MQGET calls activated cannot be predicted, because it depends on the scheduling considerations of the operating system, and other factors.)

MQGMO_WAIT is ignored if specified with MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR; no error is raised.

MQGMO_NO_WAIT

The application does not wait if no suitable message is available. This is the opposite of the MQGMO_WAIT option, and is defined to aid program documentation.

It is the default if neither is specified.

MQGMO_FAIL_IF_QUIESCING

Force the MQGET call to fail if the queue manager is in the quiescing state.

If this option is specified with MQGMO_WAIT, and the wait is outstanding at the time the queue manager enters the quiescing state, the wait is canceled and the call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING.

If MQGMO_FAIL_IF_QUIESCING is not specified and the queue manager or connection enters the quiescing state, the wait is not canceled.

The following *syncpoint options* relate to the participation of the MQGET call within a unit of work:

MQGMO_SYNCPOINT
The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

If neither this option nor MQGMO_NO_SYNCPOINT is specified, the inclusion of the get request in unit-of-work protocols is determined by the environment. On z/VSE, the get request is within the current unit of work.

MQGMO_NO_SYNCPOINT
The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.

If you specify neither this option nor MQGMO_SYNCPOINT, the inclusion of the get request in unit-of-work protocols is determined by the environment. On z/VSE, the get request is within the current unit of work.

The following *browse options* relate to browsing messages on the queue:

MQGMO_BROWSE_FIRST
When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, and positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.

An MQGET call with MQGMO_BROWSE_FIRST ignores the previous position of the browse cursor. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same Hobj handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

Specify the MQGMO_LOCK option with this option, to lock the message that is browsed.

You can specify MQGMO_BROWSE_FIRST with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. When you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that the queue manager retains for MQGET calls that remove messages from the queue. When you specify MQGMO_BROWSE_FIRST, the queue manager ignores the group and segment information for browsing, and scans the queue as though there were no current group and no current logical message. If the MQGET call is successful (completion code MQCC_OK or MQCC_WARNING), the group and segment information for browsing is set to that of the message returned; if the call fails, the group and segment information remains the same as it was before the call.

This option is not valid with any of the following options:
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_BROWSE_NEXT
MQGMO_MSG_UNDER_CURSOR
MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

MQGMO_BROWSE_NEXT
Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option.

If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by nonbrowse MQGET calls using the same Hobj handle.

Specify the MQGMO_LOCK option with this option to lock the message that is browsed.

You can specify MQGMO_BROWSE_NEXT with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. When you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle. The call fails with reason code MQRC_INCONSISTENT_BROWSE if this condition is not satisfied.

Note: Take special care when using an MQGET call to browse beyond the end of a message group (or logical message not in a group) when MQGMO_LOGICAL_ORDER is not specified. For example, if the last message in the group precedes the first message in the group on the queue, using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with MsgSeqNumber set to 1 (to find the first message of the next group) returns the first message in the group already browsed. This can happen immediately, or a number of MQGET calls later (if there are intervening groups).

To avoid the possibility of an infinite loop, open the queue twice for browse:

- Use the first handle to browse only the first message in each group.
- Use the second handle to browse only the messages within a specific group.
- Use the MQMO_* options to move the second browse cursor to the position of the first browse cursor, before browsing the messages in the group.
- Do not use MQGMO_BROWSE_NEXT to browse beyond the end of a group.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR

MQGMO_MSG_UNDER_CURSOR
MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls has been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively.

The position of the browse cursor is not changed by this call.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same Hobj handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified with MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor, so that is returned without unlocking and relocking it; the message remains locked.
- If there is no locked message, the message under the browse cursor (if there is one) is locked and returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified without MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor. This message is returned to the application and then unlocked. Because the message is now unlocked, there is no guarantee that it can be browsed again, or retrieved destructively (it can be retrieved destructively by another application getting messages from the queue).
- If there is no locked message, the message under the browse cursor (if there is one) is returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_COMPLETE_MSG is specified with MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must identify a message whose Offset field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_MSG_UNDER_CURSOR
MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

MQGMO_MSG_UNDER_CURSOR
Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR, the browse cursor must identify a message whose Offset field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

This option is not valid with any of the following options:
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_BROWSE_NEXT
MQGMO_UNLOCK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

The following *lock options* relate to locking messages on the queue:

MQGMO_LOCK
Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue.

The option can be specified only if one of the following options is also specified:
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_BROWSE_MSG_UNDER_CURSOR

Only one message can be locked for each queue handle, but this can be a logical message or a physical message:

- If you specify MQGMO_COMPLETE_MSG, all the message segments that comprise the logical message are locked to the queue handle (provided that they are all present on the queue and available for retrieval).
- If you omit MQGMO_COMPLETE_MSG, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using MQGMO_COMPLETE_MSG to retrieve or browse the logical message.

The locked message is always the one under the browse cursor, and the message can be removed from the queue by a later MQGET call that specifies the MQGMO_MSG_UNDER_CURSOR

option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If the call returns completion code MQCC_FAILED, or MQCC_WARNING with reason code MQRC_TRUNCATED_MSG_FAILED, no message is locked.

If the application does not remove the message from the queue, the lock is released by:

- Issuing another MQGET call for this handle, specifying either MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT (with or without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exceptions apply:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
  - The message is unlocked if MQCC_FAILED is returned with MQRC_NO_MSG_AVAILABLE.

  If you also specify MQGMO_LOCK, the message returned is locked. If you omit MQGMO_LOCK, there is no locked message after the call.

  If you specify MQGMO_WAIT, and no message is immediately available, the unlock on the original message occurs before the start of the wait (providing the call is otherwise free from error).
- Issuing another MQGET call for this handle, with MQGMO_BROWSE_MSG_UNDER_CURSOR (without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exception applies:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
- Issuing another MQGET call for this handle with MQGMO_UNLOCK.
- Issuing an MQCLOSE call for this handle (either explicitly, or implicitly by the application ending).

No special open option is required to specify this option, other than MQOO_BROWSE, which is needed to specify the accompanying browse option.

This option is not valid with any of the following options:
MQGMO_UNLOCK

MQGMO_UNLOCK

The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQCC_WARNING and MQRC_NO_MSG_LOCKED.

The MsgDesc, BufferLength, Buffer, and DataLength parameters are not checked or altered if you specify MQGMO_UNLOCK. No message is returned in Buffer.

No special open option is required to specify this option (although MQOO_BROWSE is needed to issue the lock request in the first place).

The following *message-data options* relate to the processing of the message data when the message is read from the queue:

MQGMO_ACCEPT_TRUNCATED_MSG
If the message buffer is too small to hold the complete message, allow the MQGET call to fill the buffer with as much of the message as the buffer can hold, issue a warning completion code, and complete its processing.

This means that:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_ACCEPTED is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold, a warning completion code is issued, but processing is not completed.

This means that:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_FAILED is returned if no other error occurs.

MQGMO_CONVERT
This option converts the application data in the message to conform to the CodedCharSetId and Encoding values specified in the MsgDesc parameter on the MQGET call, before the data is copied to the Buffer parameter.

The Format field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message.

The message data is converted by the queue manager for built-in formats, and by a user-written exit for other formats. See Chapter 7, “Message data conversion,” on page 217 for details of the data-conversion exit.

- If conversion is successful, the CodedCharSetId and Encoding fields specified in the MsgDesc parameter are unchanged on return from the MQGET call.
- If conversion fails (but the MQGET call otherwise completes without error), the message data is returned unconverted, and the CodedCharSetId and Encoding fields in MsgDesc are set to the values for the unconverted message. The completion code is MQCC_WARNING in this case.

In either case, these fields describe the character-set identifier and encoding of the message data that is returned in the Buffer parameter. See the Format field described in “MQMD – Message

descriptor" on page 774 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. See Appendix C, "Application Programming Guidance," on page 941 for more information about message grouping and segmentation.

MQGMO_LOGICAL_ORDER

This option controls the order in which messages are returned by successive MQGET calls for the queue handle. The option must be specified on each of those calls in order to have an effect.

If MQGMO_LOGICAL_ORDER is specified for successive MQGET calls for the queue handle, messages in groups are returned in the order given by their message sequence numbers, and segments of logical messages are returned in the order given by their segment offsets. This order might be different from the order in which those messages and segments occur on the queue.

Note: Specifying MQGMO_LOGICAL_ORDER has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. Thus it is perfectly safe to specify MQGMO_LOGICAL_ORDER when retrieving messages from queues that might contain a mixture of messages in groups, message segments, and unsegmented messages not in groups.

To return the messages in the required order, the queue manager retains the group and segment information between successive MQGET calls. This information identifies the current message group and current logical message for the queue handle, the current position within the group and logical message, and whether the messages are being retrieved within a unit of work. Because the queue manager retains this information, the application does not need to set the group and segment information before each MQGET call. Specifically, it means that the application does not need to set the GroupId, MsgSeqNumber, and Offset fields in MQMD. However, the application must set the MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MQMF_MSG_IN_GROUP flag is returned by the MQGET call. With MQGMO_LOGICAL_ORDER specified on successive calls, that group remains the current group until a message is returned that has:

- MQMF_LAST_MSG_IN_GROUP without MQMF_SEGMENT (that is, the last logical message in the group is not segmented), or
- MQMF_LAST_MSG_IN_GROUP with MQMF_LAST_SEGMENT (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of that MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MQMF_SEGMENT flag is returned by the MQGET call, and that logical message is terminated when the message that has the MQMF_LAST_SEGMENT flag is returned.

If no selection criteria are specified, successive MQGET calls return (in the correct order) the messages for the first message group on the queue, then the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the MatchOptions field:

MQMO_MATCH_MSG_ID
MQMO_MATCH_CORREL_ID
MQMO_MATCH_GROUP_ID

However, these options are effective only when there is no current message group or logical message; see the MatchOptions field described in "MQGMO – Get message options" on page 740 for further details.

When multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group (that is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned).

When MQGMO_LOGICAL_ORDER is specified, the MQGMO supplied on the MQGET call must not be less than MQGMO_VERSION_2, and the MQMD must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_GMO_VERSION or MQRC_WRONG_MD_VERSION, as appropriate.

If MQGMO_LOGICAL_ORDER is not specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether they are segments of logical messages. This means that messages or segments from a particular group or logical message might be returned out of order, or intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MQMO_* options specified on those calls (see the MatchOptions field described in "MQGMO – Get message options" on page 740 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the GroupId, MsgSeqNumber, Offset, and MatchOptions fields to the appropriate values, and then issue the MQGET call with MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT set, but without specifying MQGMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment

information, and subsequent MQGET calls using that queue handle can specify MQGMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

For any given queue handle, the application can mix MQGET calls that specify MQGMO_LOGICAL_ORDER with MQGET calls that do not. However, note the following points:

- If you omit MQGMO_LOGICAL_ORDER, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If you omit MQGMO_LOGICAL_ORDER, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
  MQRC_INCOMPLETE_GROUP
  MQRC_INCOMPLETE_MSG
  MQRC_INCONSISTENT_UOW

Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always MQCC_OK (assuming no other errors).

Applications that want to retrieve messages and segments in logical order are recommended to specify MQGMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQGMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. The application must then ensure that the MsgId, CorrelId, GroupId, MsgSeqNumber, and Offset fields in MQMD, and the MQMO_* options in MatchOptions in MQGMO, are set correctly, before each MQGET call.

For example, an application that wants to forward physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, must not specify MQGMO_LOGICAL_ORDER. In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By specifying neither MQGMO_LOGICAL_ORDER, nor the corresponding MQPMO_LOGICAL_ORDER on the MQPUT call, the forwarding

application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

You can specify MQGMO_LOGICAL_ORDER with any of the other MQGMO_* options, and with various of the MQMO_* options in appropriate circumstances (see above).

MQGMO_COMPLETE_MSG

Only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

Note: This is the only option that causes the queue manager to reassemble message segments. If not specified, segments are returned individually to the application if they are present on the queue (and they satisfy the other selection criteria specified on the MQGET call). Applications that do not want to receive individual segments must always specify MQGMO_COMPLETE_MSG.

To use this option, the application must provide a buffer that is big enough to accommodate the complete message, or specify the MQGMO_ACCEPT_TRUNCATED_MSG option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_COMPLETE_MSG prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the CurrentQDepth queue attribute; this means that there might be no retrievable logical messages, even though CurrentQDepth is greater than zero.

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor are the same for all segments in the logical message; usually it is only the MsgId, Offset, and MsgFlags fields that differ between segments in the logical message. However, if a segment is placed on a dead- letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the MQGMO_CONVERT option, and this can result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the segment might have a character set or encoding that differs from the other segments in the logical message when the segment arrives at the destination queue manager.

A logical message consisting of segments in which the CodedCharSetId and Encoding fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code MQCC_WARNING and reason code MQRC_INCONSISTENT_CCSIDS or MQRC_INCONSISTENT_ENCODINGS, as appropriate. This happens regardless of whether MQGMO_CONVERT is specified.

To retrieve the remaining segments, the application must reissue the MQGET call without the MQGMO_COMPLETE_MSG option, retrieving the segments one by one. MQGMO_LOGICAL_ORDER can be used to retrieve the remaining segments in order.

An application that puts segments can also set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses MQGMO_COMPLETE_MSG to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the first segment; the only exception is the MsgFlags field, which the queue manager sets to indicate that the reassembled message is the only segment.

If MQGMO_COMPLETE_MSG is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying MQGMO_COMPLETE_MSG. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options must be specified). If less than the full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must be positioned on a message whose Offset field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

MQGMO_COMPLETE_MSG implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_COMPLETE_MSG can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options apart from MQMO_MATCH_OFFSET.

MQGMO_ALL_MSGS_AVAILABLE
Messages in a group become available for retrieval only when all messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the CurrentQDepth queue attribute; this means that there may be no retrievable message groups, even though CurrentQDepth is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_MSGS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_MSGS_AVAILABLE has an effect only when there is no current group or logical message. If there is a current group or logical message, MQGMO_ALL_MSGS_AVAILABLE is ignored. This means that MQGMO_ALL_MSGS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_MSGS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_MSGS_AVAILABLE always has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying MQGMO_ALL_MSGS_AVAILABLE means that at the time that the MQGET call was issued, all the messages in the group were on the queue. However, be aware that other applications can still remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If you omit this option, messages belonging to groups can be retrieved even when the group is incomplete.

MQGMO_ALL_MSGS_AVAILABLE implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_ALL_MSGS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

MQGMO_ALL_SEGMENTS_AVAILABLE

Segments in a logical message become available for retrieval only when all segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_SEGMENTS_AVAILABLE prevents retrieval of segments belonging to incomplete logical messages. However, those segments still contribute to the value of the CurrentQDepth queue attribute; this means that there might be no retrievable logical messages, even though CurrentQDepth is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_SEGMENTS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_SEGMENTS_AVAILABLE has an effect only when there is no current logical message. If there is a current logical message, MQGMO_ALL_SEGMENTS_AVAILABLE is ignored. This means that MQGMO_ALL_SEGMENTS_AVAILABLE can remain on when processing messages in logical order.

- If MQGMO_ALL_SEGMENTS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_SEGMENTS_AVAILABLE always has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both MQGMO_COMPLETE_MSG and MQGMO_ALL_SEGMENTS_AVAILABLE require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If MQGMO_ALL_SEGMENTS_AVAILABLE is specified for a report message, the queue manager checks the queue to see if there is at least one report message for each of the segments that comprise the complete logical message. If there is, the MQGMO_ALL_SEGMENTS_AVAILABLE condition is satisfied. However, the queue manager does not check the type of the report messages present, and so there might be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of MQGMO_ALL_SEGMENTS_AVAILABLE does not imply that MQGMO_COMPLETE_MSG will succeed. If there is a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

You can specify MQGMO_ALL_SEGMENTS_AVAILABLE with any of the other MQGMO_* options, and with any of the MQMO_* options.

MQGMO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQGMO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of the Options field is MQGMO_NO_WAIT.

Reserved1 (MQCHAR)
This is a reserved field. The initial value of this field is a blank character. This field is ignored if Version is less than MQGMO_VERSION_2.

ResolvedQName (MQCHAR48)
This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This is different from the name used to open the queue if:

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Segmentation (MQCHAR)
This is a flag that indicates whether further segmentation is allowed for the message retrieved. It has one of the following values:

MQSEG_INHIBITED
Segmentation not allowed.

MQSEG_ALLOWED
Segmentation allowed.

This is an output field. The initial value of this field is MQSEG_INHIBITED. This field is ignored if Version is less than MQGMO_VERSION_2.

ReturnedLength (MQLONG)
This is an output field that the queue manager sets to the length in bytes of the message data returned by the MQGET call in the Buffer parameter. If the queue manager does not support this capability, ReturnedLength is set to the value MQRL_UNDEFINED.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the MQGET call:

- If ReturnedLength is not MQRL_UNDEFINED, the number of bytes of message data returned is given by ReturnedLength.
- If ReturnedLength has the value MQRL_UNDEFINED, the number of bytes of message data returned is usually given by the smaller of BufferLength and DataLength, but can be less than this if the MQGET call completes with reason code MQRC_TRUNCATED_MSG_ACCEPTED. If this happens, the insignificant bytes in the Buffer parameter are set to nulls.

This special value is defined:

MQRL_UNDEFINED
Length of returned data not defined.

The initial value of this field is MQRL_UNDEFINED. This field is ignored if Version is less than MQGMO_VERSION_3.

SegmentStatus (MQCHAR)
This is a flag that indicates whether the message retrieved is a segment of a logical message. It has one of the following values:

MQSS_NOT_A_SEGMENT
Message is not a segment.

MQSS_SEGMENT
Message is a segment, but is not the last segment of the logical message.

MQSS_LAST_SEGMENT
Message is the last segment of the logical message.

This is also the value returned if the logical message consists of only one segment.

This is an output field. The initial value of this field is MQSS_NOT_A_SEGMENT.

This field is ignored if Version is less than MQGMO_VERSION_2.

Signal1 (MQLONG)
This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option; it identifies a signal that is to be delivered when a message is available.

This field is not supported by WebSphere MQ for z/VSE.

Signal2 (MQLONG)
This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option. It is a reserved field; its value is not significant. The initial value of this field is 0.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQGMO_STRUC_ID
Identifier for get-message options structure. For the C programming language, the constant MQGMO_STRUC_ID_ARRAY is also defined; this has the same value as MQGMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQGMO_STRUC_ID.

Version (MQLONG)
This is the structure version number. The value must be one of the following:

MQGMO_VERSION_1
Version-1 get-message options structure. This version is supported in all environments.

MQGMO_VERSION_2
Version-2 get-message options structure. This version is supported in all environments.

MQGMO_VERSION_3
Version-3 get-message options structure. This version is supported in all environments.

MQGMO_VERSION_4
Version-4 get-message options structure. This version is supported in all environments.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields.

This is always an input field. The initial value of this field is MQGMO_VERSION_1.

WaitInterval (MQLONG)
This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the MsgDesc parameter of the MQGET call; see the MsgId field described in “MQMD – Message descriptor” on page 774 for more details.

If no suitable message has arrived after this time has elapsed, the call completes with MQCC_FAILED and reason code MQRC_NO_MSG_AVAILABLE.

WaitInterval is used in conjunction with the MQGMO_WAIT option. It is ignored if MQGMO_WAIT is not specified. If it is specified, WaitInterval must be greater than or equal to zero, or the following special value:

MQWI_UNLIMITED
Unlimited wait interval.

The initial value of this field is 0.

C declaration

```
typedef struct tagMQGMO {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    Options;        /* Options that control the action of
                               MQGET */
  MQLONG    WaitInterval;   /* Wait interval */
  MQLONG    Signal1;        /* Signal */
  MQLONG    Signal2;        /* Reserved */
  MQCHAR48  ResolvedQName;  /* Resolved name of destination queue */
  /* Ver:1 */
  MQLONG    MatchOptions;   /* Options controlling selection criteria
                               used for MQGET */
  MQCHAR    GroupStatus;    /* Flag indicating whether message
                               retrieved is in a group */
  MQCHAR    SegmentStatus;  /* Flag indicating whether message
                               retrieved is a segment of a logical
                               message */
  MQCHAR    Segmentation;   /* Flag indicating whether segmentation is
                               allowed for the message retrieved */
  MQCHAR    Reserved1;      /* Reserved */
  /* Ver:2 */
  MQBYTE16  MsgToken;       /* Message token                          */
  MQLONG    ReturnedLength; /* Length of message data returned
                                (bytes)                               */
  /* Ver:3 */
  MQLONG              Reserved2;     /* Reserved                     */
  MQHMSG              MsgHandle;     /* Message handle               */
  /* Ver:4 */
 } MQGMO;
```

COBOL declaration

```
**   MQGMO structure
  10 MQGMO.
**    Structure identifier
   15 MQGMO-STRUCID       PIC X(4).
**    Structure version number
   15 MQGMO-VERSION       PIC S9(9) BINARY.
**    Options
   15 MQGMO-OPTIONS       PIC S9(9) BINARY.
**    Wait interval
   15 MQGMO-WAITINTERVAL  PIC S9(9) BINARY.
**    Signal
   15 MQGMO-SIGNAL1       PIC S9(9) BINARY.
**    Reserved
   15 MQGMO-SIGNAL2       PIC S9(9) BINARY.
**    Resolved name of destination queue
   15 MQGMO-RESOLVEDQNAME PIC X(48).
** Ver:1 **
**    Options controlling selection criteria used for MQGET
   15 MQGMO-MATCHOPTIONS  PIC S9(9) BINARY.
**    Flag indicating whether message retrieved is in a group
   15 MQGMO-GROUPSTATUS   PIC X.
**    Flag indicating whether message retrieved is a segment of a
**    logical message
   15 MQGMO-SEGMENTSTATUS PIC X.
**    Flag indicating whether further segmentation is allowed for
**    the message retrieved
   15 MQGMO-SEGMENTATION  PIC X.
**    Reserved
   15 MQGMO-RESERVED1     PIC X.
** Ver:2 **
**    Message token
   15 MQGMO-MSGTOKEN       PIC X(16).
**    Length of message data returned (bytes)
   15 MQGMO-RETURNEDLENGTH PIC S9(9) BINARY.
** Ver:3 **
**    Reserved
   15 MQGMO-RESERVED2      PIC S9(9) BINARY.
**    Message handle
   15 MQGMO-MSGHANDLE      PIC S9(18) BINARY.
** Ver:4 **
```

PL/I declaration

```
dcl
 1 MQGMO based,
  3 StrucId       char(4)
      init(MQGMO_STRUC_ID),      /* Structure identifier */
  3 Version       fixed bin(31)
      init(MQGMO_VERSION_1),     /* Structure version number */
  3 Options       fixed bin(31)
      init(MQGMO_NO_WAIT),       /* Options that control the action of
                                    MQGET */
  3 WaitInterval  fixed bin(31)
      init(0),                   /* Wait interval */
  3 Signal1       pointer
      init(null()),              /* Pointer to signal */
  3 Signal2       fixed bin(31)
      init(0),                   /* Signal identifier */
  3 ResolvedQName char(48)
      init(''),                  /* Resolved name of destination
                                    queue */
  /* Ver:1 */
  3 MatchOptions  fixed bin(31)
   init((MQMO_MATCH_MSG_ID+MQMO_MATCH_CORREL_ID)),  /* Options
                                 controlling selection criteria
                                 used for MQGET */
  3 GroupStatus  char(1)
   init(MQGS_NOT_IN_GROUP),      /* Flag indicating whether message
                                    retrieved is in a group */
  3 SegmentStatus  char(1)
   init(MQSS_NOT_A_SEGMENT),     /* Flag indicating whether message
                                    retrieved is a segment of a logical
                                    message */
  3 Segmentation  char(1)
   init(MQSEG_INHIBITED),        /* Flag indicating whether further
                                    segmentation is allowed for the
                                    message retrieved */
  3 Reserved1  char(1)
   init(''),                     /* Reserved */
  /* Ver:2 */
  3 MsgToken     char(16)
    init(MQMTOK_NONE),                 /* Message token */
  3 ReturnedLength    fixed bin(31)
    init(MQRL_UNDEFINED),              /* Length of message data */
                                       /* returned (bytes) */
  /* Ver:3 */
  3 Reserved2         fixed bin(31)
    init(0),                           /* Reserved */
  3 MsgHandle         char(8)
    init(MQHM_NONE);                   /* Message handle */
  /* Ver:4 */
```

MQIMPO – Inquire message property options

The MQIMPO structure allows applications to specify options that control how properties of messages are inquired. The structure is an input parameter on the MQINQMP call.

Data in MQIMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 45. Fields in MQIMPO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQIMPO
RequestedEncoding	Encoding into which the enquired property is to be converted

Table 45. Fields in MQIMPO (continued)

Field	Description
RequestedCCSID	Character set of the inquired property
ReturnedEncoding	Encoding of the returned value
ReturnedCCSID	Character set of returned value
Reserved1	Reserved field
ReturnedName	Name of the inquired property
TypeString	String representation of the data type of the property

Here is a description of the fields.

Options (MQLONG)

The options shown here control the action of MQINQMP. You can specify one or more of these options and, if you need more than one, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Value data options:

The following options relate to the processing of the value data when the property is retrieved from the message.

MQIMPO_CONVERT_VALUE

This option requests that the value of the property be converted to conform to the RequestedCCSID and RequestedEncoding values specified before the MQINQMP call returns the property value in the Value area.

If conversion is successful, the ReturnedCCSID and ReturnedEncoding fields are set to the same as RequestedCCSID and RequestedEncoding on return from the MQINQMP call. If conversion fails, but the MQINQMP call otherwise completes without error, the property value is returned unconverted.

If the property is a string, the ReturnedCCSID and ReturnedEncoding fields are set to the character set and encoding of the unconverted string. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_VALUE_NOT_CONVERTED. The property cursor is advanced to the returned property.

If the property value expands during conversion and exceeds the size of the Value parameter, the value is returned unconverted with completion code MQCC_FAILED; the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The DataLength parameter of the MQINQMP call returns the length that the property value would have converted to, in order

to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

This option also requests that:

- If the property name contains a wildcard, and The ReturnedName field is initialized with an address or offset for the returned name, then the returned name is converted to conform to the RequestedCCSID and RequestedEncoding values.
- If conversion is successful, the VSCCSID field of ReturnedName and the encoding of the returned name are set to the input value of RequestedCCSID and RequestedEncoding.
- If conversion fails, but the MQINQMP call otherwise completes without error or warning, the returned name is unconverted. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_NAME_NOT_CONVERTED.

The property cursor is advanced to the returned property. MQRC_PROP_VALUE_NOT_CONVERTED is returned if both the value and the name are not converted.

If the returned name expands during conversion, and exceeds the size of the VSBufsize field of the RequestedName, the returned string is left unconverted with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_NAME_TOO_BIG.

The VSLength field of the MQCHARV structure returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

MQIMPO_CONVERT_TYPE

This option requests that the value of the property be converted from its current data type into the data type specified on the Type parameter of the MQINQMP call.

- If conversion is successful, the Type parameter is unchanged on return of the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The property cursor is unchanged.

If the conversion of the data type causes the value to expand during conversion and the converted value exceeds the size of the Value parameter, the value is returned unconverted with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The DataLength parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

If the value of the Type parameter of the MQINQMP call is not valid, the call fails with reason MQRC_PROPERTY_TYPE_ERROR. If the requested data type conversion is not supported, the call fails

with reason MQRC_PROP_CONV_NOT_SUPPORTED. Table 46 shows the data type conversions that are supported.

Table 46. Data type conversions supported by MQIMPO

Property data type	Supported target data types
MQTYPE_BOOLEAN	MQTYPE_STRING, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32
MQTYPE_BYTE_STRING	MQTYPE_STRING
MQTYPE_INT8	MQTYPE_STRING, MQTYPE_INT16, MQTYPE_INT32
MQTYPE_INT16	MQTYPE_STRING, MQTYPE_INT32
MQTYPE_INT32	MQTYPE_STRING
MQTYPE_FLOAT32	MQTYPE_STRING
MQTYPE_FLOAT64	MQTYPE_STRING
MQTYPE_STRING	MQTYPE_BOOLEAN, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_FLOAT32, MQTYPE_FLOAT64
MQTYPE_NULL	None

The general rules governing the supported conversions are:

- Numeric property values can be converted from one data type to another, provided that no data is lost during the conversion.

  For example, the value of a property with data type MQTYPE_INT16 can be converted into a value with data type MQTYPE_INT32, but cannot be converted into a value with data type MQTYPE_INT8.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- If an application attempts a conversion that is not supported, WebSphere MQ returns reason code MQRC_PROP_CONV_NOT_SUPPORTED.

The specific rules for converting a property value from one data type to another are as follows:

- When converting an MQTYPE_BOOLEAN property value to a string, the value TRUE is converted to the string "TRUE", and the value false is converted to the string "FALSE".
- When converting an MQTYPE_BOOLEAN property value to a numeric data type, the value TRUE is converted to one, and the value FALSE is converted to zero.
- When converting a string property value to an MQTYPE_BOOLEAN value, the string "TRUE" , or "1" , is converted to TRUE, and the string "FALSE", or "0", is converted to FALSE.

Note: The terms "TRUE" and "FALSE" are not case-sensitive. Any other string cannot be converted. WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.

- When converting a string property value to a value with data type MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, or MQTYPE_INT64, the string must have the format:

  [blanks][sign]digits

  The meanings of the components of the string are:

  blanks
  Optional leading blank characters.

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

  WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly. When converting a string property value to a value with data type MQTYPE_FLOAT32 or MQTYPE_FLOAT64, the string must have the format:

  [blanks][sign]digits[.digits][e_char[e_sign]e_digits]

  The meanings of the components of the string are :

  blanks
  Optional leading blank characters.

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  e_char An exponent character, which is either "E" or "e".

  e_sign An optional plus sign (+) or minus sign (-) character for the exponent.

  e_digits
  A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

  After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

  WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.
- When converting a numeric property value to a string, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte string property value to a string, each byte is converted to the two hexadecimal characters that

represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

MQIMPO_QUERY_LENGTH
Query the type and length of the property value. The length is returned in the DataLength parameter of the MQINQMP call. The property value is not returned.

If a ReturnedName buffer is specified, the VSLength field of the MQCHARV structure is filled in with the length of the property name. The property name is not returned.

Iteration options:

The following options relate to iterating over properties, using a name with a wildcard character.

MQIMPO_INQ_FIRST
Inquire on the first property that matches the specified name. After this call, a cursor is established on the property that is returned. This is the default value.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call, if required, to inquire on the same property again. Note that there is only one property cursor. Therefore, if the property name specified in the MQINQMP call changes, the cursor is reset.

This option is not valid with either of these options:
MQIMPO_INQ_NEXT
MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_NEXT
Inquires on the next property that matches the specified name, continuing the search from the property cursor. The cursor is advanced to the property that is returned.

If this is the first MQINQMP call for the specified name, then the first property that matches the specified name is returned.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call if required, to inquire on the same property again.

If the property under the cursor has been deleted, MQINQMP returns the next matching property following the one that has been deleted.

If a property is added that matches the wildcard while an iteration is in progress, the property may be returned during the completion of the iteration. The property is returned once the iteration restarts using MQIMPO_INQ_FIRST.

A property matching the wildcard that was deleted while the iteration was in progress, is not returned subsequent to its deletion. This option is not valid with either of these options:
MQIMPO_INQ_FIRST
MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_PROP_UNDER_CURSOR
Retrieve the value of the property pointed to by the property

cursor. The property pointed to by the property cursor is the one that was last inquired, using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, when the message handle is specified in the MsgHandle field of the MQGMO on an MQGET call, or when the message handle is specified in OriginalMsgHandle or NewMsgHandle fields of the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

This option is not valid with either of these:
MQIMPO_INQ_FIRST
MQIMPO_INQ_NEXT

If none of the options previously described is required, the following option can be used:

MQIMPO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values.

MQIMPO_NONE aids program documentation; it is not intended that this option be used with any other option but, as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is MQIMPO_INQ_FIRST.

RequestedCCSID (MQLONG)
The character set that the inquired property value is to be converted into if the value is a character string. This is also the character set into which the ReturnedName is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQCCSI_APPL.

RequestedEncoding (MQLONG)
This is the encoding into which the inquired property value is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQENC_NATIVE.

Reserved1 (MQCHAR)
This is a reserved field. The initial value of this field is a blank character.

ReturnedCCSID (MQLONG)
On output, this is the character set of the value returned if the Type parameter of the MQINQMP call is MQTYPE_STRING.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the ReturnedCCSID field, on return, is the same value as the value passed in.

The initial value of this field is zero.

ReturnedEncoding (MQLONG)
On output, this is the encoding of the value returned.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the ReturnedEncoding field, on return, is the same value as the value passed in.

The initial value of this field is MQENC_NATIVE.

ReturnedName (MQCHARV)
The actual name of the inquired property.

On input, a string buffer can be passed in using the VSPtr or VSOffset field of the MQCHARV structure. The length of the string buffer is specified using the VSBufsize field of the MQCHARV structure.

On return from the MQINQMP call, the string buffer is completed with the name of the property that was inquired, provided the string buffer was long enough to fully contain the name. The VSLength field of the MQCHARV structure is filled in with the length of the property name. The VSCCSID field of the MQCHARV structure is filled in to indicate the character set of the returned name, whether or not conversion of the name failed.

This is an input/output field. The initial value of this field is MQCHARV_DEFAULT.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQIMPO_STRUC_ID
Identifier for buffer to message handle structure.

For the C programming language, the constant MQIMPO_STRUC_ID_ARRAY is also defined; this has the same value as MQIMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQIMPO_STRUC_ID.

TypeString (MQCHAR8)
A string representation of the data type of the property.

If the property was specified in an MQRFH2 header and the MQRFH2 dt attribute is not recognized, this field can be used to determine the data type of the property. TypeString is returned in coded character set 1208 (UTF-8), and is the first eight bytes of the value of the dt attribute of the property that failed to be recognized.

This is always an output field. The initial value of this field is the null string in the C programming language, and 8 blank characters in other programming languages.

Version (MQLONG)
This is the structure version number.

The value must be:

MQIMPO_VERSION_1
Version number for inquire message property options structure.

The following constant specifies the version number of the current version:

MQIMPO_CURRENT_VERSION
Current version of inquire message property options structure.

This is always an input field. The initial value of this field is MQIMPO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQIMPO.

Table 47. Initial values of fields in MQIMPO

Field name	Name of constant	Value of constant
StrucId	MQIMPO_STRUC_ID	'IMPO'
Version	MQIMPO_VERSION_1	1
Options	MQIMPO_INQ_FIRST	
RequestedEncoding	MQENC_NATIVE	
RequestedCCSID	MQCCSI_APPL	
ReturnedEncoding	MQENC_NATIVE	
ReturnedCCSID	0	
Reserved1	0	
ReturnedName	MQCHARV_DEFAULT	
TypeString	Null string or blanks	

Note:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQIMPO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   MQIMPO MyIMPO = {MQIMPO_DEFAULT};

C declaration

```
struct tagMQIMPO {
  MQCHAR4  StrucId;            /* Structure identifier */
  MQLONG   Version;            /* Structure version number */
  MQLONG   Options;            /* Options that control the action of
                                  MQINQMP */
  MQLONG   RequestedEncoding;  /* Requested encoding of Value */
  MQLONG   RequestedCCSID;     /* Requested character set identifier
                                  of Value */
  MQLONG   ReturnedEncoding;   /* Returned encoding of Value */
  MQLONG   ReturnedCCSID;      /* Returned character set identifier of
                                  Value */
  MQLONG   Reserved1;          /* Reserved */
  MQCHARV  ReturnedName;       /* Returned property name */
  MQCHAR8  TypeString;         /* Property data type as a string */
};
```

COBOL declaration

```
** MQIMPO structure
 10  MQIMPO.
** Structure identifier
  15  MQIMPO-STRUCID                    PIC X(4).
** Structure version number
  15  MQIMPO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQINQMP
  15  MQIMPO-OPTIONS                    PIC S9(9) BINARY.
** Requested encoding of Value
  15  MQIMPO-REQUESTEDENCODING          PIC S9(9) BINARY.
** Requested character set identifier of Value
  15  MQIMPO-REQUESTEDCCSID             PIC S9(9) BINARY.
** Returned encoding of Value
  15  MQIMPO-RETURNEDENCODING           PIC S9(9) BINARY.
** Returned character set identifier of Value
  15  MQIMPO-RETURNEDCCSID              PIC S9(9) BINARY.
** Reserved
  15  MQIMPO-RESERVED1                  PIC S9(9) BINARY.
** Returned property name
  15  MQIMPO-RETURNEDNAME.
** Address of variable length string
   20  MQIMPO-RETURNEDNAME-VSPTR        USAGE POINTER.
** Offset of variable length string
   20  MQIMPO-RETURNEDNAME-VSOFFSET     PIC S9(9) BINARY.
** Size of buffer
   20  MQIMPO-RETURNEDNAME-VSBUFSIZE    PIC S9(9) BINARY.
** Length of variable length string
   20  MQIMPO-RETURNEDNAME-VSLENGTH     PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQIMPO-RETURNEDNAME-VSCCSID      PIC S9(9) BINARY.
** Property data type as a string
  15  MQIMPO-TYPESTRING PIC             X(8).
```

PL/I declaration

```
dcl
 1 MQIMPO based,
  3 StrucId          char(4)
    init(MQIMPO_STRUC_ID),          /* Structure identifier */
  3 Version          fixed bin(31)
    init(MQIMPO_VERSION_1),         /* Structure version number */
  3 Options          fixed bin(31)
    init(MQIMPO_INQ_FIRST),         /* Options that control the */
                                    /* action of MQINQMP */
  3 RequestedEncoding fixed bin(31)
    init(MQENC_NATIVE),             /* Requested encoding of Value */
  3 RequestedCCSID   fixed bin(31)
    init(MQCCSI_APPL),              /* Requested character set */
                                    /* identifier of Value */
  3 ReturnedEncoding fixed bin(31)
    init(MQENC_NATIVE),             /* Returned encoding of Value */
  3 ReturnedCCSID    fixed bin(31)
    init(0),                        /* Returned character set */
                                    /* identifier of Value */
  3 Reserved1        fixed bin(31)
    init(0),                        /* Reserved */
  3 ReturnedName,                   /* Returned property name */
    5 VSPtr          pointer
      init(null()),                 /* Address of variable length */
                                    /* string */
    5 VSOffset       fixed bin(31)
      init(0),                      /* Offset of variable length */
                                    /* string */
    5 VSBufSize      fixed bin(31)
      init(0),                      /* Size of buffer */
    5 VSLength       fixed bin(31)
      init(0),                      /* Length of variable length */
                                    /* string */
    5 VSCCSID        fixed bin(31)
      init(MQCCSI_APPL),            /* CCSID of variable length */
                                    /* string */
  3 TypeString       char(8)
    init('');                       /* Property data type as a */
```

MQMD – Message descriptor

The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT, and MQPUT1 calls.

Data in MQMD must be in the character set and encoding of the local queue manager; these are given by the CodedCharSetId queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD can be converted automatically. It is not necessary for the application to convert the MQMD.

A version-2 MQMD is generally equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described below.

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the Format field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

  Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.
- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a non-default value. The Format field in MQMD will have the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

Certain fields in MQMD contain the message context. There are two types of message context: identity context and origin context. Usually:

- Identity context relates to the application that originally put the message.
- Origin context relates to the application that most recently put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Fields

Here is a summary of the fields.

Table 48. Fields in MQMD

Field	Description
StrucId	Structure identifier

Table 48. Fields in MQMD (continued)

Field	Description
Version	Structure version number
Report	Options for report messages
MsgType	Message type
Expiry	Message lifetime
Feedback	Feedback or reason code
Encoding	Numeric encoding of message data
CodedCharSetId	Character set identifier of message data
Format	Format name of message data
Priority	Message priority
Persistence	Message persistence
MsgId	Message identifier
CorrelId	Correlation identifier
BackoutCount	Backout counter
ReplyToQ	Name of reply queue
ReplyToQMgr	Name of reply queue manager
UserIdentifier	User identifier
AccountingToken	Accounting token
ApplIdentityData	Application data relating to identity
PutApplType	Type of application that put the message
PutApplName	Name of application that put the message
PutDate	Date when message was put
PutTime	Time when message was put
ApplOriginData	Application data relating to origin
Note: The remaining fields are ignored if Version is less than MQMD_VERSION_2.	
GroupId	Group identifier
MsgSeqNumber	Sequence number of logical message within group
Offset	Offset of data in physical message from start of logical message
MsgFlags	Message flags
OriginalLength	Length of original message

Here is a description of the fields.

AccountingToken (MQBYTE32)

This is the accounting token, part of the identity context of the message. AccountingToken allows an application to charge appropriately for work done as a result of the message. The queue manager treats this information as a string of bits and does not check its content.

WebSphere MQ for z/VSE does not support this field.

ApplIdentityData (MQCHAR32)
This is part of the identity context of the message. ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator.

WebSphere MQ for z/VSE does not support this field.

ApplOriginData (MQCHAR4)
This is part of the origin context of the message. ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message.

For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted. The queue manager treats this information as character data, but does not define the format of it.

When the queue manager generates this information, it is entirely blank.

WebSphere MQ for z/VSE does not support this field.

BackoutCount (MQLONG)
This is a count of the number of times that the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It helps the application to detect processing errors that are based on message content. The count excludes MQGET calls that specify any of the MQGMO_BROWSE_* options.

WebSphere MQ for z/VSE does not support this field.

CodedCharSetId (MQLONG)
This specifies the character set identifier of character data in the message.

Note: Character data in MQMD and the other MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager's CodedCharSetId attribute; see “Global QUEUE /File Names” on page 85 for details of this attribute.
You can use the following special values:

MQCCSI_Q_MGR
Character data in the message is in the queue manager's character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD that is sent with the message to the true character-set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCCSI_DEFAULT
The CodedCharSetId of the data in the String field is defined by the CodedCharSetId field in the header structure that precedes the MQCFH structure, or by the CodedCharSetId field in the MQMD if the MQCFH is at the start of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the value MQCCSI_Q_MGR in the MQMD that is sent with the message as described above, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified. Applications that retrieve messages must compare this field against the value the application is expecting; if the values differ, the application might need to convert character data in the message.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the coded character-set identifier to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value MQCCSI_Q_MGR is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character-set identifier of the unconverted message that is returned to the application. Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls.

The initial value of this field is MQCCSI_Q_MGR.

CorrelId (MQBYTE24)

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies MQPMO_NEW_CORREL_ID, the queue manager generates a unique correlation identifier which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

When the queue manager or a message channel agent generates a report message, it sets the CorrelId field in the way specified by the Report field of the original message, either MQRO_COPY_MSG_ID_TO_CORREL_ID or MQRO_PASS_CORREL_ID. Applications that generate report messages must also do this.

For the MQGET call, CorrelId is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the MsgId field for details on how to specify values for this field.

Specifying MQCI_NONE as the correlation identifier has the same effect as not specifying MQMO_MATCH_CORREL_ID, that is, any correlation identifier will match. If the MQGMO_MSG_UNDER_CURSOR option is specified in the GetMsgOpts parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the CorrelId field is set to the correlation identifier of the message returned (if any).

The following special values can be used:

MQCI_NONE

No correlation identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQCI_NONE_ARRAY is also defined; this has the same value as MQCI_NONE, but is an array of characters instead of a string.

MQCI_NEW_SESSION
Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

For the C programming language, the constant MQCI_NEW_SESSION_ARRAY is also defined; this has the same value as MQCI_NEW_SESSION, but is an array of characters instead of a string.

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if MQPMO_NEW_CORREL_ID is not specified, and an output field if MQPMO_NEW_CORREL_ID is specified. The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

Encoding (MQLONG)
This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid.

The following special value is defined:

MQENC_NATIVE
The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages usually specify MQENC_NATIVE. Applications that retrieve messages must compare this field against the value MQENC_NATIVE; if the values differ, the application might need to convert numeric data in the message. Use the MQGMO_CONVERT option to request the queue manager to convert the message as part of the processing of the MQGET call.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQENC_NATIVE.

Expiry (MQLONG)
This is a period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

Message expiry is described in detail in "Message expiry" on page 165.

The following special value is recognized:

MQEI_UNLIMITED
The message has an unlimited expiration time.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQEI_UNLIMITED.

Feedback (MQLONG)
This is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message. The field can contain one of the MQFB_* values, or one of the MQRC_* values. Feedback codes are grouped as follows:

MQFB_NONE
No feedback provided.

MQFB_SYSTEM_FIRST
Lowest value for system-generated feedback.

MQFB_SYSTEM_LAST
Highest value for system-generated feedback.

The range of system-generated feedback codes MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST includes the general feedback codes listed below (MQFB_*), and also the reason codes (MQRC_*) that can occur when the message cannot be put on the destination queue.

MQFB_APPL_FIRST
Lowest value for application-generated feedback.

MQFB_APPL_LAST
Highest value for application-generated feedback.

Applications that generate report messages must not use feedback codes in the system range (other than MQFB_QUIT), unless they want to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be MQFB_NONE, or be within the system range or application range. This is checked whatever the value of MsgType.

General feedback codes:

MQFB_COA
Confirmation of arrival on the destination queue (see MQRO_COA).

MQFB_COD
Confirmation of delivery to the receiving application (see MQRO_COD).

MQFB_EXPIRATION
Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

MQFB_PAN
Positive action notification (see MQRO_PAN).

MQFB_NAN
Negative action notification (see MQRO_NAN).

MQFB_QUIT
End application.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

CICS-bridge feedback codes: The following feedback codes can be generated by the CICS bridge:

MQFB_CICS_APPL_ABENDED
The application program specified in the message abended. This feedback code occurs only in the Reason field of the MQDLH structure.

MQFB_CICS_APPL_NOT_STARTED
The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the Reason field of the MQDLH structure.

MQFB_CICS_BRIDGE_FAILURE
CICS bridge terminated abnormally without completing normal error processing.

MQFB_CICS_CCSID_ERROR
Character set identifier not valid.

MQFB_CICS_CIH_ERROR
CICS information header structure missing or not valid.

MQFB_CICS_COMMAREA_ERROR
Length of CICS commarea not valid.

MQFB_CICS_CORREL_ID_ERROR
Correlation identifier not valid.

MQFB_CICS_DLQ_ERROR
The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

MQFB_CICS_ENCODING_ERROR
Encoding not valid.

MQFB_CICS_INTERNAL_ERROR
CICS bridge encountered an unexpected error. This feedback code occurs only in the Reason field of the MQDLH structure.

MQFB_CICS_NOT_AUTHORIZED
User identifier not authorized or password not valid. This feedback code occurs only in the Reason field of the MQDLH structure.

MQFB_CICS_UOW_BACKED_OUT
The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

MQFB_CICS_UOW_ERROR
Unit-of-work control field UOWControl not valid.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQFB_NONE.

Format (MQCHAR8)
This is a name that the sender of the message uses to indicate to the receiver the nature of the data in the message. Any characters that are in the queue manager's character set can be specified for the name, but it is recommended that the name be restricted to the following:

- Uppercase A through Z.
- Numeric digits 0 through 9.

If other characters are used, it might not be possible to translate the name between the character sets of the sending and receiving queue managers.

Pad the name with blanks to the length of the field, or use a null character to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described above.

Names beginning MQ in upper, lower, and mixed case have meanings that are defined by the queue manager; do not use names beginning with these letters for your own formats. The queue manager built-in formats are:

MQFMT_NONE
The nature of the data is undefined: the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

If you specify MQGMO_CONVERT on the MQGET call, and the character set or encoding of data in the message differs from that specified in the MsgDesc parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR if the MQFMT_NONE data is at the beginning of the message.
- Completion code MQCC_OK and reason code MQRC_NONE if the MQFMT_NONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

For the C programming language, the constant MQFMT_NONE_ARRAY is also defined; this has the same value as MQFMT_NONE, but is an array of characters instead of a string.

MQFMT_ADMIN
The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. See Chapter 8, “Programmable system

management," on page 221 for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_ADMIN_ARRAY is also defined; this has the same value as MQFMT_ADMIN, but is an array of characters instead of a string.

MQFMT_CICS
The message data begins with the CICS information header MQCIH, followed by the application data. The format name of the application data is given by the Format field in the MQCIH structure.

Specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_CICS.

For the C programming language, the constant MQFMT_CICS_ARRAY is also defined; this has the same value as MQFMT_CICS, but is an array of characters instead of a string.

MQFMT_COMMAND_1
The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_1_ARRAY is also defined; this has the same value as MQFMT_COMMAND_1, but is an array of characters instead of a string.

MQFMT_COMMAND_2
The message is an MQSC command-server reply message containing information about the objects requested. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_2_ARRAY is also defined; this has the same value as MQFMT_COMMAND_2, but is an array of characters instead of a string.

MQFMT_DEAD_LETTER_HEADER
The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the Format field in the MQDLH structure; see "MQDLH – Dead-letter header" on page 726 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

COA and COD reports are not generated for messages that have a Format of MQFMT_DEAD_LETTER_HEADER.

For the C programming language, the constant MQFMT_DEAD_LETTER_HEADER_ARRAY is also defined; this has the same value as MQFMT_DEAD_LETTER_HEADER, but is an array of characters instead of a string.

MQFMT_DIST_HEADER
The message data begins with the distribution-list header MQDH;

this includes the arrays of MQOR and MQPMR records. The distribution- list header can be followed by additional data. Messages with format MQFMT_DIST_HEADER can be converted if the MQGMO_CONVERT option is specified on the MQGET call. This format is supported in the following environments: AIX, HP-UX, i5/OS™, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_DIST_HEADER_ARRAY is also defined; this has the same value as MQFMT_DIST_HEADER, but is an array of characters instead of a string.

MQFMT_EVENT
The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands; refer to Chapter 8, "Programmable system management," on page 221 for more information about this structure, and for information about events.

Version-1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call.

Version-2 event messages can be converted only on z/OS.

For the C programming language, the constant MQFMT_EVENT_ARRAY is also defined; this has the same value as MQFMT_EVENT, but is an array of characters instead of a string.

MQFMT_IMS
The message data begins with the IMS™ information header MQIIH, which is followed by the application data. Specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_IMS.

For the C programming language, the constant MQFMT_IMS_ARRAY is also defined; this has the same value as MQFMT_IMS, but is an array of characters instead of a string.

MQFMT_IMS_VAR_STRING
The message is an IMS variable string, which is a string of the form *llzzccc*, where:

ll Is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of ll (2 bytes), plus the length of *zz* (2 bytes), plus the length of the character string itself. *ll* is a 2-byte binary integer in the encoding specified by the Encoding field.

zz Is a 2-byte field containing flags that are significant to IMS. *zz* is a byte string consisting of two MQBYTE fields, and is transmitted without change from sender to receiver (that is, *zz* is not subject to any conversion).

ccc Is a variable-length character string containing *ll*-4 characters. *ccc* is in the character set specified by the CodedCharSetId field.

On z/OS, the message data can consist of a sequence of IMS variable strings butted together, with each string being of the form

llzzccc. There must be no bytes skipped between successive IMS variable strings. This means that if the first string has an odd length, the second string will be misaligned, that is, it will not begin on a boundary that is a multiple of two. Take care when constructing such strings on machines that require alignment of elementary data types.

Use the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_IMS_VAR_STRING.

For the C programming language, the constant MQFMT_IMS_VAR_STRING_ARRAY is also defined; this has the same value as MQFMT_IMS_VAR_STRING, but is an array of characters instead of a string.

MQFMT_MD_EXTENSION
The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data that follow the MQMDE are given by the Format, CodedCharSetId, and Encoding fields in the MQMDE. See "MQMDE – Message descriptor extension" on page 806 for details of this structure.

Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_MD_EXTENSION_ARRAY is also defined; this has the same value as MQFMT_MD_EXTENSION, but is an array of characters instead of a string.

MQFMT_PCF
The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Refer to Chapter 8, "Programmable system management," on page 221 for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_PCF_ARRAY is also defined; this has the same value as MQFMT_PCF, but is an array of characters instead of a string.

MQFMT_REF_MSG_HEADER
The message data begins with the reference message header MQRMH, and is optionally followed by other data. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_REF_MSG_HEADER_ARRAY is also defined; this has the same value as MQFMT_REF_MSG_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER
The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER_2
The message data begins with the version-2 rules and formatting header MQRFH2, and is optionally followed by other data. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_2_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER_2, but is an array of characters instead of a string.

MQFMT_STRING
The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_STRING_ARRAY is also defined; this has the same value as MQFMT_STRING, but is an array of characters instead of a string.

MQFMT_TRIGGER
The message is a trigger message, described by the MQTM structure; see "MQTM – Trigger message" on page 863 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_TRIGGER_ARRAY is also defined; this has the same value as MQFMT_TRIGGER, but is an array of characters instead of a string.

MQFMT_XMIT_Q_HEADER
The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the Format field in the MQMD structure, which is part of the transmission queue header MQXQH. See "MQXQH – Transmission-queue header" on page 866 for details of this structure.

COA and COD reports are not generated for messages that have a Format of MQFMT_XMIT_Q_HEADER.

For the C programming language, the constant MQFMT_XMIT_Q_HEADER_ARRAY is also defined; this has the same value as MQFMT_XMIT_Q_HEADER, but is an array of characters instead of a string.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_FORMAT_LENGTH.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24)

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. GroupId is also used if segmentation is allowed for the message. In all these cases, GroupId has a non-null value, and one or more of the following flags is set in the MsgFlags field:

MQMF_MSG_IN_GROUP
MQMF_LAST_MSG_IN_GROUP
MQMF_SEGMENT
MQMF_LAST_SEGMENT
MQMF_SEGMENTATION_ALLOWED

If none of these flags is set, GroupId has the special null value MQGI_NONE.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_GROUP_ID is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that GroupId is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, applications must not generate their own group identifiers; instead, applications must do one of the following:

- If MQPMO_LOGICAL_ORDER is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. This is the recommended procedure.
- If MQPMO_LOGICAL_ORDER is not specified, the application must request the queue manager to generate the group identifier, by setting GroupId to MQGI_NONE on the first MQPUT or MQPUT1 call for a message in the group or segment of the logical message. The group identifier returned by the queue manager on output from that call must then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

  When MQPMO_LOGICAL_ORDER is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the first MQPUT or MQPUT1 call that is issued for any of those messages.

On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message if the object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application

needs to know the group identifiers generated, the application must provide MQPMR records containing the GroupId field.

On output from the MQGET call, the queue manager sets this field to the value for the message retrieved. The following special value is defined:

MQGI_NONE
No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

For the C programming language, the constant MQGI_NONE_ARRAY is also defined; this has the same value as MQGI_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_GROUP_ID_LENGTH. The initial value of this field is MQGI_NONE. This field is ignored if Version is less than MQMD_VERSION_2.

MsgFlags (MQLONG)
These are flags that specify attributes of the message, or control its processing. The flags are divided into the following categories:
- Segmentation flags.
- Status flags.

Segmentation flags
When a message is too big for a queue, an attempt to put the message on the queue usually fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application that retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message that is returned by the MQGET call. The latter is achieved by specifying the MQGMO_COMPLETE_MSG option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See "MQGMO – Get message options" on page 740 for details of the MQGMO_COMPLETE_MSG option.)

A message can be segmented at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MQMF_SEGMENTATION_INHIBITED
This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments. The value of this flag is binary zero. This is the default.

MQMF_SEGMENTATION_ALLOWED
This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments.

MQMF_SEGMENTATION_ALLOWED can be set without either MQMF_SEGMENT or MQMF_LAST_SEGMENT being set.

When the queue manager segments a message, the queue manager turns on the MQMF_SEGMENT flag in the copy of the MQMD that is sent with each segment, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call. For the last segment in the logical message, the queue manager also turns on the MQMF_LAST_SEGMENT flag in the MQMD that is sent with the segment.

Note: Take care when putting messages with MQMF_SEGMENTATION_ALLOWED but without MQPMO_LOGICAL_ORDER. If the message is:
- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must reset the GroupId field to MQGI_NONE before each MQPUT or MQPUT1 call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the GroupId field and the MQPMO_LOGICAL_ORDER option for more information about when to reset the GroupId field.

The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager (see below), and only the last segment created from a message can be smaller than this limit. (The lower limit for the size of an application-generated segment is one byte.) Segments generated by the queue manager might be of unequal length. The queue-manager processes the message as follows:

- User-defined formats are split on boundaries that are multiples of 16 bytes; the queue manager does not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than MQFMT_STRING are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes. The second or later segment generated by the queue manager begins with one of the following:
  - An MQ header structure.
  - The start of the application message data.
  - Part of the way through the application message data.
- MQFMT_STRING is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this might result in segments that cannot be converted from one character set to another (see below). The queue manager never splits

MQFMT_STRING messages into segments that are smaller than 16 bytes (other than the last segment).

- The queue manager sets the Format, CodedCharSetId, and Encoding fields in the MQMD of each segment to describe correctly the data present at the start of the segment; the format name is either the name of a built-in format, or the name of a user-defined format.
- The Report field in the MQMD of segments with Offset greater than zero is modified. For each report type, if the report option is MQRO_*_WITH_DATA, but the segment cannot contain any of the first 100 bytes of user data (that is, the data following any MQ header structures that may be present), the report option is changed to MQRO_*.

The queue manager follows the above rules, but otherwise splits messages as it thinks fit; you cannot assume that the queue manager splits a message in a particular way.

Take special care when converting data in messages that might be segmented:

- If the receiving application converts data on the MQGET call, and specifies the MQGMO_COMPLETE_MSG option, the data-conversion exit is passed the complete message for the exit to convert, and the fact that the message was segmented is be apparent to the exit.
- If the receiving application retrieves one segment at a time, the data-conversion exit is invoked to convert one segment at a time. The exit must therefore be capable of converting the data in a segment independently of the data in any of the other segments. If the nature of the data in the message is such that arbitrary segmentation of the data on 16-byte boundaries might result in segments that cannot be converted by the exit, or the format is MQFMT_STRING and the character set is DBCS or mixed SBCS/DBCS, the sending application must create and put the segments, specifying MQMF_SEGMENTATION_INHIBITED to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.
- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages that are not segments of logical messages; the MCA never attempts to convert messages that are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the Segmentation field in MQGMO.

The initial value of this flag is MQMF_SEGMENTATION_INHIBITED.

Status flags

These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MQMF_MSG_IN_GROUP
Message is a member of a group.

MQMF_LAST_MSG_IN_GROUP
Message is the last logical message in a group. If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MQMF_LAST_MSG_IN_GROUP is set, but the MsgSeqNumber field has the value one.

MQMF_SEGMENT
Message is a segment of a logical message. When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (excluding the lengths of any MQ header structures that might be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

MQMF_LAST_SEGMENT
Message is the last segment of a logical message. If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

A logical message can consist of only one segment. If this is the case, MQMF_LAST_SEGMENT is set, but the Offset field has the value zero.

When MQMF_LAST_SEGMENT is specified, the length of the application message data in the segment (excluding the lengths of any header structures that might be present) can be zero.

The application must ensure that these flags are set correctly when putting messages. If MQPMO_LOGICAL_ORDER is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to successive MQPUT calls for the queue handle when MQPMO_LOGICAL_ORDER is specified:

- If there is no current group or logical message, all these flags (and combinations of them) are valid.
- Once MQMF_MSG_IN_GROUP has been specified, it must remain on until MQMF_LAST_MSG_IN_GROUP is specified. The call fails with reason code MQRC_INCOMPLETE_GROUP if this condition is not satisfied.

- Once MQMF_SEGMENT has been specified, it must remain on until MQMF_LAST_SEGMENT is specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified without MQMF_MSG_IN_GROUP, MQMF_MSG_IN_GROUP must remain off until after MQMF_LAST_SEGMENT has been specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the GroupStatus and SegmentStatus fields in MQGMO.

Default flags
The following can be specified to indicate that the message has default attributes:

MQMF_NONE
No message flags (default message attributes). This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQMF_NONE. This field is ignored if Version is less than MQMD_VERSION_2.

MsgId (MQBYTE24)
This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MQMI_NONE or MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier when the message is put, and places it in the message descriptor sent with the message. The queue manager also returns this message identifier in the message descriptor belonging to the sending application. The application can use this value to record information about particular messages, and to respond to queries from other parts of the application.

The sending application can also specify a value for the message identifier other than MQMI_NONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this to propagate the message identifier of the original message.

The queue manager does not use this field except to:
- Generate a unique value if requested, as described above.
- Deliver the value to the application that issues the get request for the message.

- Copy the value to the CorrelId field of any report message that it generates about this message (depending on the Report options).

When the queue manager or a message channel agent generates a report message, it sets the MsgId field in the way specified by the Report field of the original message, either MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID. Applications that generate report messages must also do this.

For the MQGET call, MsgId is one of the five fields that can be used to retrieve a particular message from the queue. Normally the MQGET call returns the next message on the queue, but a particular message can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

MsgId
CorrelId
GroupId
MsgSeqNumber
Offset

The application sets one or more of these field to the values required, and then sets the corresponding MQMO_* match options in the MatchOptions field in MQGMO to use those fields as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the MatchOptions field (if not altered by the application) is to match both the message identifier and the correlation identifier.

Normally, the message returned is the first message on the queue that satisfies the selection criteria. But if MQGMO_BROWSE_NEXT is specified, the message returned is the next message that satisfies the selection criteria; the scan for this message starts with the message following the current cursor position.

Note: The queue is scanned sequentially for a message that satisfies the selection criteria, so retrieval times are slower than if no selection criteria are specified, especially if many messages have to be scanned before a suitable one is found.

Specifying MQMI_NONE as the message identifier has the same effect as not specifying MQMO_MATCH_MSG_ID, that is, any message identifier matches.

This field is ignored if the MQGMO_MSG_UNDER_CURSOR option is specified in the GetMsgOpts parameter on the MQGET call.

On return from an MQGET call, the MsgId field is set to the message identifier of the message returned (if any).

The following special value can be used:

MQMI_NONE
No message identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQMI_NONE_ARRAY is also defined; this has the same value as MQMI_NONE, but is an array of characters instead of a string.

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by MQ_MSG_ID_LENGTH. The initial value of this field is MQMI_NONE.

MsgSeqNumber (MQLONG)
This is the sequence number of a logical message within a group.Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message that is not in a group has a sequence number of 1.

The application does not have to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_MSG_SEQ_NUMBER is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that MsgSeqNumber is set to an appropriate value.

On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

On output from the MQGET call, the queue manager sets this field to the value for the message retrieved. The initial value of this field is one. This field is ignored if Version is less than MQMD_VERSION_2.

MsgType (MQLONG)
This indicates the type of the message. Message types are grouped as follows:

MQMT_SYSTEM_FIRST
Lowest value for system-defined message types.

MQMT_SYSTEM_LAST
Highest value for system-defined message types.

The following values are currently defined within the system range:

MQMT_DATAGRAM
The message is one that does not require a reply.

MQMT_REQUEST
The message is one that requires a reply.

Specify the name of the queue to which to send the reply in the ReplyToQ field. The Report field indicates how to set the MsgId and CorrelId of the reply.

MQMT_REPLY
The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the ReplyToQ field of the request message. Use the Report field of the request to control how to set the MsgId and CorrelId of the reply.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MQMT_REPORT
The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Send the message to the queue indicated by the ReplyToQ field of the message descriptor of the original message. Set the Feedback field s to indicate the nature of the report. Use the Report field of the original message to control how to set the MsgId and CorrelId of the report message.

Report messages generated by the queue manager or message channel agent are always sent to the ReplyToQ queue, with the Feedback and CorrelId fields set as described above.

Application-defined values can also be used. They must be within the following range:

MQMT_APPL_FIRST
Lowest value for application-defined message types.

MQMT_APPL_LAST
Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the MsgType value must be within either the system-defined range or the application- defined range; if it is not, the call fails with reason code MQRC_MSG_TYPE_ERROR.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQMT_DATAGRAM.

Offset (MQLONG)
This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a segment. The offset is in the range 0 through 999 999 999. A physical message that is not a segment of a logical message has an offset of zero.

The application does not need to set this field on the MQPUT or MQGET call if:
- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_OFFSET is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that Offset is set to an appropriate value.

On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message. For a report message reporting on a segment of a logical message, the OriginalLength field (provided it is not MQOL_UNDEFINED) is used to update the offset in the segment information retained by the queue manager.

On output from the MQGET call, the queue manager sets this field to the value for the message retrieved. The initial value of this field is zero. This field is ignored if Version is less than MQMD_VERSION_2.

OriginalLength (MQLONG)
This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report message the GroupId, MsgSeqNumber, Offset, and MsgFlags, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages must do the same, and set the OriginalLength field correctly.

The following special value is defined:

MQOL_UNDEFINED
Original length of message not defined.

OriginalLength is an input field on the MQPUT and MQPUT1 calls, but the value that the application provides is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
  - Greater than zero if the segment is not the last segment.
  - Not less than zero if the segment is the last segment.
  - Not less than the length of data present in the message.

  If these conditions are not satisfied, the call fails with reason code MQRC_ORIGINAL_LENGTH_ERROR.
- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value MQOL_UNDEFINED instead.

This is an output field on the MQGET call. The initial value of this field is MQOL_UNDEFINED. This field is ignored if Version is less than MQMD_VERSION_2.

Persistence (MQLONG)
This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT
The message survives system failures and restarts of the queue manager. Once the message has been put, and the unit of work in which it was put has been committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the unit of work in which it was got has been committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism holds the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on temporary dynamic queues.

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT
The message does not usually survive system failures or queue

manager restarts. This applies even if an intact copy of the message is found on auxiliary storage when the queue manager restarts.

For WebSphere MQ for z/VSE, non-persistent messages can only be placed on temporary dynamic queues.

MQPER_PERSISTENCE_AS_Q_DEF
For WebSphere MQ for z/VSE, this value is converted by the queue manager to MQPER_PERSISTENT for predefined and permanent dynamic queues, and to MQPER_NOT_PERSISTENT for temporary dynamic queues.

When replying to a message, applications must use the persistence of the request message for the reply message.

For an MQGET call, the value returned is either MQPER_PERSISTENT or MQPER_NO_PERSISTENT.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPER_PERSISTENCE_AS_Q_DEF.

Priority (MQLONG)
The priority of the message.

This field is not supported by WebSphere MQ for z/VSE, but if it is set, and the message's destination is to another system, WebSphere MQ for z/VSE will preserve the value.

PutApplName (MQCHAR28)
This is the name of application that put the message, and is part of the origin context of the message. The format of the PutApplName depends on the value of PutApplType.

When the queue manager sets this field, it sets the field to a value that is determined by the environment. WebSphere MQ for z/VSE sets this field to the CICS application id concatenated with the CICS transaction name.

PutApplType (MQLONG)
This is the type of application that put the message, and is part of the origin context of the message.

PutApplType can have one of a number of standard types. You can also define your own types, but only with values in the range MQAT_USER_FIRST through MQAT_USER_LAST.

WebSphere MQ for z/VSE uses the standard type:

MQAT_CICS_VSE
CICS for z/VSE transaction.

PutDate (MQCHAR8)
This is the date when the message was put, and is part of the origin context of the message.

The format used for the date when this field is generated by the queue manager is YYYYMMDD, where the characters represent:

YYYY Year (four numeric digits)

MM Month of year (01 through 12)

DD Day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the PutDate and PutTime fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the PutDate that was transmitted with the message.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8)

This is the time when the message was put, and is part of the origin context of the message. The format used for the time when this field is generated by the queue manager is HHMMSSTH, where the characters represent (in order):

HH Hours (00 through 23)

MM Minutes (00 through 59)

SS Seconds (00 through 59; see note below)

T Tenths of a second (0 through 9)

H Hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in PutTime. This happens when leap seconds are inserted into the global time standard.
Greenwich Mean Time (GMT) is used for the PutDate and PutTime fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

The queue manager does not check the contents of the field, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the PutTime that was transmitted with the message.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

ReplyToQ (MQCHAR48)

This is the name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by ReplyToQMgr. This queue must not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the MsgType field has the value MQMT_REQUEST, or if any report messages are requested by the Report field. However, the value specified (or substituted; see below) is passed on to the application that issues the get request for the message, whatever the message type.

If the ReplyToQMgr field is blank, the local queue manager looks up the ReplyToQ name in its own queue definitions. If a local definition of a remote queue exists with this name, the ReplyToQ value in the transmitted message is replaced by the value of the RemoteQName attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, ReplyToQ is unchanged.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the ReplyToQ is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, set the ReplyToQ field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReplyToQMgr (MQCHAR48)

This is the name of the queue manager to which to send the reply message or report message. ReplyToQ is the local name of a queue that is defined on this queue manager.

If the ReplyToQMgr field is blank, the local queue manager looks up the ReplyToQ name in its queue definitions. If a local definition of a remote queue exists with this name, the ReplyToQMgr value in the transmitted message is replaced by the value of the RemoteQMgrName attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the ReplyToQMgr that is transmitted with the message is the name of the local queue manager.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the ReplyToQMgr is replaced in the transmitted message.

If a reply-to queue is not required, set the ReplyToQMgr field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Report (MQLONG)

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The Report field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

Exception.
Expiration.
Confirm on arrival (COA).
Confirm on delivery (COD).

If more than one type of report message is required, or other report options are needed, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

The application that receives the report message can determine the reason that the report was generated by examining the Feedback field in the MQMD; see the Feedback field for more details.

Exception options

Specify one of the options listed below to request an exception report message.

MQRO_EXCEPTION

A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

Message data from the original message is not included with the report message. Do not specify more than one of

MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_DATA
This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_FULL_DATA
Exception reports with full data required.

This is the same as MQRO_EXCEPTION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

Expiration options: Specify one of the options listed below to request an expiration report message.

MQRO_EXPIRATION
This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed (see the Expiry field). If this option is not set, no report message is generated if a message is discarded for this reason (even if you specify one of the MQRO_EXCEPTION_* options).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA
This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA
This is the same as MQRO_EXPIRATION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

Confirm-on-arrival options

Specify one of the options listed below to request a confirm-on-arrival report message.

MQRO_COA

This type of report is generated by the queue manager that owns the destination queue when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager can be retrieved only if the unit of work is committed.

A COA report is not generated if the Format field in the message descriptor is MQFMT_XMIT_Q_HEADER or MQFMT_DEAD_LETTER_HEADER. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA

This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA

This is the same as MQRO_COA, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

Confirm-on-delivery options

Specify one of the options listed below to request a confirm-on-delivery report message.

MQRO_COD

This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not generated if the Format field in the message descriptor is MQFMT_DEAD_LETTER_HEADER. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA
This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

If MQGMO_ACCEPT_TRUNCATED_MSG is specified on the MQGET call for the original message, and the message retrieved is truncated, the amount of application message data placed in the report message depends on the environment:

- On z/OS, it is the minimum of:
  - The length of the original message.
  - The length of the buffer used to retrieve the message.
  - 100 bytes.
- In other environments, it is the minimum of:
  - The length of the original message.
  - 100 bytes.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA
This is the same as MQRO_COD, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

Default option
Specify the following if no report options are required:

MQRO_NONE
Use this value to indicate that no other options have been specified. MQRO_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQRO_NONE.

StrucId (MQCHAR4)
This is the structure identifier, and must be: MQMD_STRUC_ID Identifier for message descriptor structure.

For the C programming language, the constant MQMD_STRUC_ID_ARRAY is also defined; this has the same value as MQMD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMD_STRUC_ID.

UserIdentifier (MQCHAR12)
This is part of the identity context of the message.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

WebSphere MQ for z/VSE populates this field with the user id associated with the CICS transaction.

Version (MQLONG)
This is the structure version number, and must be one of the following:

MQMD_VERSION_1
Version-1 message descriptor structure. This version is supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure. This version is supported in all WebSphere MQ V6 environments, plus WebSphere MQ clients connected to these systems, and WebSphere MQ for z/VSE.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields.

C declaration

MQMD - Message descriptor

```
typedef struct tagMQMD MQMD;
struct tagMQMD
{
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Report;          /* Options for report messages */
  MQLONG    MsgType;         /* Message type */
  MQLONG    Expiry;          /* Message lifetime */
  MQLONG    Feedback;        /* Feedback or reason code */
  MQLONG    Encoding;        /* Numeric encoding of message data */
  MQLONG    CodedCharSetId;  /* Character set identifier of message data */
  MQCHAR8   Format;          /* Format name of message data */
  MQLONG    Priority;        /* Message priority */
  MQLONG    Persistence;     /* Message persistence */
  MQBYTE24  MsgId;           /* Message identifier */
  MQBYTE24  CorrelId;        /* Correlation identifier */
  MQLONG    BackoutCount;    /* Backout counter */
  MQCHAR48  ReplyToQ;        /* Name of reply queue */
  MQCHAR48  ReplyToQMgr;     /* Name of reply queue manager */
  MQCHAR12  UserIdentifier;  /* User identifier */
  MQBYTE32  AccountingToken; /* Accounting token */
  MQCHAR32  ApplIdentityData; /* Application data relating to identity */
  MQLONG    PutApplType;     /* Type of application that put the message */
  MQCHAR28  PutApplName;     /* Name of application that put the message */
  MQCHAR8   PutDate;         /* Date when message was put */
  MQCHAR8   PutTime;         /* Time when message was put */
  MQCHAR4   ApplOriginData;  /* Application data relating to origin */
  MQBYTE24  GroupId;         /* Group identifier */
  MQLONG    MsgSeqNumber;    /* Sequence number of logical message within group */
  MQLONG    Offset;          /* Offset of data in physical msg from start of logical msg */
  MQLONG    MsgFlags;        /* Message flags */
  MQLONG    OriginalLength;  /* Length of original message */
};
```

COBOL declaration

```
** MQMD structure
 10 MQMD.
** Structure identifier
  15 MQMD-STRUCID PIC X(4).
** Structure version number
  15 MQMD-VERSION PIC S9(9) BINARY.
** Options for report messages
  15 MQMD-REPORT PIC S9(9) BINARY.
** Message type
  15 MQMD-MSGTYPE PIC S9(9) BINARY.
** Message lifetime
  15 MQMD-EXPIRY PIC S9(9) BINARY.
** Feedback or reason code
  15 MQMD-FEEDBACK PIC S9(9) BINARY.
** Numeric encoding of message data
  15 MQMD-ENCODING PIC S9(9) BINARY.
** Character set identifier of message data
  15 MQMD-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of message data
  15 MQMD-FORMAT PIC X(8).
** Message priority
  15 MQMD-PRIORITY PIC S9(9) BINARY.
** Message persistence
  15 MQMD-PERSISTENCE PIC S9(9) BINARY.
** Message identifier
  15 MQMD-MSGID PIC X(24).
** Correlation identifier
  15 MQMD-CORRELID PIC X(24).
** Backout counter
  15 MQMD-BACKOUTCOUNT PIC S9(9) BINARY.
** Name of reply queue
  15 MQMD-REPLYTOQ PIC X(48).
** Name of reply queue manager
  15 MQMD-REPLYTOQMGR PIC X(48).
** User identifier
  15 MQMD-USERIDENTIFIER PIC X(12).
** Accounting token
  15 MQMD-ACCOUNTINGTOKEN PIC X(32).
** Application data relating to identity
  15 MQMD-APPLIDENTITYDATA PIC X(32).
** Type of application that put the message
  15 MQMD-PUTAPPLTYPE PIC S9(9) BINARY.
** Name of application that put the message
  15 MQMD-PUTAPPLNAME PIC X(28).
** Date when message was put
  15 MQMD-PUTDATE PIC X(8).
** Time when message was put
  15 MQMD-PUTTIME PIC X(8).
** Application data relating to origin
  15 MQMD-APPLORIGINDATA PIC X(4).
** Group identifier
  15 MQMD-GROUPID PIC X(24).
** Sequence number of logical message within group
  15 MQMD-MSGSEQNUMBER PIC S9(9) BINARY.
** Offset of data in physical message from start of logical message
  15 MQMD-OFFSET PIC S9(9) BINARY.
** Message flags
  15 MQMD-MSGFLAGS PIC S9(9) BINARY.
** Length of original message
  15 MQMD-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQMD based,
    3 StrucId char(4),             /* Structure identifier */
  3 Version fixed bin(31),         /* Structure version number */
  3 Report fixed bin(31),          /* Options for report messages */
  3 MsgType fixed bin(31),         /* Message type */
  3 Expiry fixed bin(31),          /* Message lifetime */
  3 Feedback fixed bin(31),        /* Feedback or reason code */
  3 Encoding fixed bin(31),        /* Numeric encoding of message data */
  3 CodedCharSetId fixed bin(31),  /* Character set identifier of message data */
  3 Format char(8),                /* Format name of message data */
  3 Priority fixed bin(31),        /* Message priority */
  3 Persistence fixed bin(31),     /* Message persistence */
  3 MsgId char(24),                /* Message identifier */
  3 CorrelId char(24),             /* Correlation identifier */
  3 BackoutCount fixed bin(31),    /* Backout counter */
  3 ReplyToQ char(48),             /* Name of reply queue */
  3 ReplyToQMgr char(48),          /* Name of reply queue manager */
  3 UserIdentifier char(12),       /* User identifier */
  3 AccountingToken char(32),      /* Accounting token */
  3 ApplIdentityData char(32),     /* Application data relating to identity */
  3 PutApplType fixed bin(31),     /* Type of application that put the message */
  3 PutApplName char(28),          /* Name of application that put the message */
  3 PutDate char(8),               /* Date when message was put */
  3 PutTime char(8),               /* Time when message was put */
  3 ApplOriginData char(4),        /* Application data relating to origin */
  3 GroupId char(24),              /* Group identifier */
  3 MsgSeqNumber fixed bin(31),    /* Sequence number of logical message within group */
  3 Offset fixed bin(31),          /* Offset of data in physical msg from start of logical msg */
  3 MsgFlags fixed bin(31),        /* Message flags */
  3 OriginalLength fixed bin(31);  /* Length of original message */
```

MQMDE – Message descriptor extension

The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Applications that use a version-2 MQMD will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, might encounter an MQMDE in some situations.

The MQMDE structure can occur in the following circumstances:
- Specified on the MQPUT and MQPUT1 calls.
- Returned by the MQGET call.
- In messages on transmission queues.

MQMDE specified on MQPUT and MQPUT1 calls

On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the Format field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

If the application provides a version-2 MQMD and prefixes the application message data with an MQMDE, the structure is treated as message data.

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MQMF_SEGMENT or MQMF_LAST_SEGMENT flag is set), and the format name in the MQMD is MQFMT_DEAD_LETTER_HEADER, the queue manager generates an MQMDE structure and inserts it between the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call

On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the Format field in MQMD to the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the Buffer parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If the MQGET call returns an MQMDE, the data in the MQMDE is usually in the queue manager's character set and encoding.

MQMDE in messages on transmission queues

Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE might also be present, positioned between the MQXQH structure and application message data, but it is usually present only if one or more of the fields in the MQMDE has a nondefault value.

Other MQ header structures can also occur between the MQXQH structure and the application message data. For example, when the dead-letter header MQDLH is present, and the message is not a segment, the order is:

- MQXQH (containing a version-1 MQMD)
- MQMDE
- MQDLH
- Application message data

Fields

Here is a summary of the fields.

Table 49. Fields in MQMDE

Field	Description
StrucId	Structure identifier
Version	Structure version number
StrucLength	Length of MQMDE structure
Encoding	Numeric encoding of data that follows MQMDE
CodedCharSetId	Character set identifier of data that follows MQMDE
Format	Format name of data that follows MQMDE
Flags	General flags

Table 49. Fields in MQMDE (continued)

Field	Description
GroupId	Group identifier
MsgSeqNumber	Sequence number of logical message within group
Offset	Offset of data in physical message from start of logical message
MsgFlags	Message flags
OriginalLength	Length of original message

Here is a description of the fields.

CodedCharSetId (MQLONG)
This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

MQCCSI_INHERIT
Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)
This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the Encoding field described in "MQMD – Message descriptor" on page 774 for more information about data encodings.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)
The following flag can be specified:

MQMDEF_NONE
No flags.

The initial value of this field is MQMDEF_NONE.

Format (MQCHAR8)
This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the Format field described in "MQMD – Message descriptor" on page 774 for more information about format names.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24)
See the GroupId field described in "MQMD – Message descriptor" on page 774.

The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG)
See the MsgFlags field described in "MQMD – Message descriptor" on page 774.

The initial value of this field is MQMF_NONE.

MsgSeqNumber (MQLONG)
See the MsgSeqNumber field described in "MQMD – Message descriptor" on page 774.

The initial value of this field is 1.

Offset (MQLONG)
See the Offset field described in "MQMD – Message descriptor" on page 774.

The initial value of this field is 0.

OriginalLength (MQLONG)
See the OriginalLength field described in "MQMD – Message descriptor" on page 774.

The initial value of this field is MQOL_UNDEFINED.

StrucId (MQCHAR4)
The value must be:

MQMDE_STRUC_ID
Identifier for message descriptor extension structure.

For the C programming language, the constant MQMDE_STRUC_ID_ARRAY is also defined; this has the same value as MQMDE_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQMDE_STRUC_ID.

StrucLength (MQLONG)
This is the length of the MQMDE structure; the following value is defined:

MQMDE_LENGTH_2
Length of version-2 message descriptor extension structure.

The initial value of this field is MQMDE_LENGTH_2.

Version (MQLONG)
This is the structure version number; the value must be:

MQMDE_VERSION_2
Version-2 message descriptor extension structure.

The following constant specifies the version number of the current version:

MQMDE_CURRENT_VERSION
Current version of message descriptor extension structure.

The initial value of this field is MQMDE_VERSION_2.

C declaration

```
typedef struct tagMQMDE MQMDE;
struct tagMQMDE
{
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQMDE structure */
  MQLONG    Encoding;        /* Numeric encoding of data that follows MQMDE */
  MQLONG    CodedCharSetId;  /* Character-set identifier of data that follows MQMDE */
  MQCHAR8   Format;          /* Format name of data that follows MQMDE */
  MQLONG    Flags;           /* General flags */
  MQBYTE24  GroupId;         /* Group identifier */
  MQLONG    MsgSeqNumber;    /* Sequence number of logical message within group */
  MQLONG    Offset;          /* Offset of data in physical msg from start of logical msg */
  MQLONG    MsgFlags;        /* Message flags */
  MQLONG    OriginalLength   /* Length of original message */
};
```

COBOL declaration

```
  ** MQMDE structure
   10 MQMDE.
  ** Structure identifier
    15 MQMDE-STRUCID PIC X(4).
  ** Structure version number
    15 MQMDE-VERSION PIC S9(9) BINARY.
  ** Length of MQMDE structure
    15 MQMDE-STRUCLENGTH PIC S9(9) BINARY.
  ** Numeric encoding of data that follows MQMDE
    15 MQMDE-ENCODING PIC S9(9) BINARY.
  ** Character-set identifier of data that follows MQMDE
    15 MQMDE-CODEDCHARSETID PIC S9(9) BINARY.
  ** Format name of data that follows MQMDE
    15 MQMDE-FORMAT PIC X(8).
  ** General flags
    15 MQMDE-FLAGS PIC S9(9) BINARY.
  ** Group identifier
    15 MQMDE-GROUPID PIC X(24).
  ** Sequence number of logical message within group
    15 MQMDE-MSGSEQNUMBER PIC S9(9) BINARY.
  ** Offset of data in physical message from start of logical message
    15 MQMDE-OFFSET PIC S9(9) BINARY.
  ** Message flags
    15 MQMDE-MSGFLAGS PIC S9(9) BINARY.
  ** Length of original message
    15 MQMDE-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration

```
  dcl
   1 MQMDE based,
     3 StrucId char(4),               /* Structure identifier */
     3 Version fixed bin(31),         /* Structure version number */
     3 StrucLength fixed bin(31),     /* Length of MQMDE structure */
     3 Encoding fixed bin(31),        /* Numeric encoding of data that follows MQMDE */
     3 CodedCharSetId fixed bin(31),  /* Character-set identifier of data that follows MQMDE */
     3 Format char(8),                /* Format name of data that follows MQMDE */
     3 Flags fixed bin(31),           /* General flags */
     3 GroupId char(24),              /* Group identifier */
     3 MsgSeqNumber fixed bin(31),    /* Sequence number of logical message within group */
     3 Offset fixed bin(31),          /* Offset of data in physical msg from start of logical msg */
     3 MsgFlags fixed bin(31),        /* Message flags */
     3 OriginalLength fixed bin(31);  /* Length of original message */
```

MQMHBO – Message handle to buffer options

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles. The structure is an input parameter on the MQMHBUF call.

Data in MQMHBO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 50. Fields in MQMHBO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQMHBO

Here is a description of the fields.

Options (MQLONG)
These options control the action of MQMHBUF.

You must specify the following option:

MQMHBO_PROPERTIES_IN_MQRFH2
When converting properties from a message handle into a buffer, convert them into the MQRFH2 format.

Optionally, you can also specify the following value. If required, values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQMHBO_DELETE_PROPERTIES
Properties that are added to the buffer are deleted from the message handle. If the call fails no properties are deleted.

This is always an input field. The initial value of this field is MQMHBO_PROPERTIES_IN_MQRFH2.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQMHBO_STRUC_ID
Identifier for buffer to message handle structure.

For the C programming language, the constant MQMHBO_STRUC_ID_ARRAY is also defined. This has the same value as MQMHBO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMHBO_STRUC_ID.

Version (MQLONG)
This is the structure version number.

The value must be:

MQMHBO_VERSION_1
Version number for message handle to buffer options structure.

The following constant specifies the version number of the current version:

MQMHBO_CURRENT_VERSION
Current version of message handle to buffer options structure.

This is always an input field. The initial value of this field is MQMHBO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQMHBO.

Table 51. Initial values of fields in MQMHBO

Field name	Name of constant	Value of constant
StrucId	MQMHBO_STRUC_ID	'MHBO'
Version	MQMHBO_VERSION_1	1
Options	MQMHBO_NONE	0

Note:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMHBO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   MQMHBO MyMHBO = {MQMHBO_DEFAULT};

C declaration

```
struct tagMQMHBO {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQMHBUF */
};
```

COBOL declaration

```
** MQBMHO structure
 10  MQBMHO.
** Structure identifier
 15  MQBMHO-STRUCID                    PIC X(4).
** Structure version number
 15  MQBMHO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQBUFMH
 15  MQBMHO-OPTIONS                    PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQMHBO based,
  3 StrucId             char(4)
    init(MQMHBO_STRUC_ID),             /* Structure identifier */
  3 Version             fixed bin(31)
    init(MQMHBO_VERSION_1),            /* Structure version number */
  3 Options             fixed bin(31)
    init(MQMHBO_PROPERTIES_IN_MQRFH2); /* Options that control the */
                                       /* action of MQMHBUF */
```

MQOD – Object descriptor

The MQOD structure is used to specify an object by name. The following types of object are valid with WebSphere MQ for z/VSE:

- Queue
- Queue manager
- Namelist

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

Fields

Here is a summary of the fields.

Table 52. Fields in MQOD

Field	Description
StrucId	Structure identifier
Version	Structure version number
ObjectType	Object type
ObjectName	Object name
ObjectQMgrName	Object queue manager name
DynamicQName	Dynamic queue name
AlternateUserId	Alternate user identifier
RecsPresent	Number of object records present
KnownDestCount	Number of local queues opened successfully
UnknownDestCount	Number of remote queues opened successfully
InvalidDestCount	Number of queues that failed to open
ObjectRecOffset	Offset of first object record from start of MQOD
ResponseRecOffset	Offset of first response record from start of MQOD
ObjectRecPtr	Address of first object record
ResponseRecPtr	Address of first response record

Here is a description of the fields.

AlternateUserId (MQCHAR12)
This field contains an alternate user identifier that is used to check the authorization for the open, in place of the user identifier that the application is currently running under.

This field is not supported by WebSphere MQ for z/VSE.

DynamicQName (MQCHAR48)
Dynamic queue name.

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when ObjectName specifies the name of a model queue; in all other cases DynamicQName is ignored.

The characters that are valid in the name are the same as those for ObjectName (see above), except that an asterisk is also valid (see below). A name that is completely blank (or one in which only blanks appear before the first null character) is not valid if ObjectName is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to appear in the first character position, in which case the name consists solely of the characters generated by the queue manager.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment. On WebSphere MQ for z/VSE, the value is 'AMQ.*'.

The value is a blank-padded string.

InvalidDestCount (MQLONG)
Number of queues that failed to open.

This is the number of queues in the distribution list that failed to open successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

Note: If present, this field is set only if the CompCode parameter on the MQOPEN or MQPUT1 call is MQCC_OK or MQC_WARNING; it is not set if the CompCode parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

KnownDestCount (MQLONG)
Number of local queues opened successfully.

This is the number of queues in the distribution list that resolve to local queues and that were opened successfully. The count does not include queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

ObjectName (MQCHAR48)
This is the local name of the object as defined on the queue manager identified by ObjectQMgrName. The name can contain the following characters:
- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.)
- Forward slash (/)

- Underscore (_)
- Percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks.

The following points apply to the types of object indicated:

- If ObjectName is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the ObjectName field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If ObjectType is MQOT_Q_MGR, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

This is an input/output field for the MQOPEN call when ObjectName is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)

This is the name of the queue manager on which the ObjectName object is defined. The characters that are valid in the name are the same as those for ObjectName. A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If ObjectType is MQOT_Q_MGR, ObjectQMgrName must be blank or the name of the local queue manager.
- If ObjectName is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the ObjectQMgrName field the name of the queue manager on which the queue is created; this is the name of the local queue manager. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.

This is an input/output field for the MQOPEN call when ObjectName is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectRecOffset (MQLONG)

Offset of first object record from start of MQOD..

This is the offset in bytes of the first MQOR object record from the start of the MQOD structure. The offset can be positive or negative. ObjectRecOffset is used only when a distribution list is being opened. The field is ignored if RecsPresent is zero.

When a distribution list is being opened, an array of one or more MQOR object records must be provided in order to specify the names of the destination queues in th distribution list. This can be done in one or two ways:

- By using the offset field ObjectRecOffset

In this case, the application should declare its own structure containing an MQOD, followed by the array of MQOR records (with as many array elements as are needed), and set ObjectRecOffset to the offset of the first element in the array from the start of the MQOD. Care must be taken to ensure that this offset is correct.

Using ObjectOffset is recommended for programming languages which do not support the pointer data type, or which implement the pointer data type in a fashion which is not portable to different environments, as in the COBOL programming language.

- By using the pointer field ObjectRecPtr

  In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set ObjectRecPtr to the address of the array.

  Using ObjectRecPtr is recommended fro programming languages which support the pointer data type in a fashion which is portable to different environments, as in the C programming language.

Whichever technique is chosen, one of ObjectRecOffset and ObjectRecPtr must be used; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are non zero.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

ObjectRecPtr (MQLONG)
Address of first object record.

This is the address of the first MQOR object record. ObjectRecPtr is used only when a distribution list is being opened. The field is ignored if ObjectRecPresent is zero.

Either ObjectRecPtr or ObjectRecOffset can be used to specify the object records, but not both; see the description of the ObjectRecOffset field above for details. If ObjectRecPtr is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and all-null byte string otherwise. This field is ignored if Version is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer date type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string

ObjectType (MQLONG)
The type of object being named in ObjectName. Possible values are:

MQOT_Q
Queue

MQOT_Q_MGR
Queue manager

This is always an input field. The initial value of this field is MQOT_Q.

RecsPresent (MQLONG)
Number of object records present.

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with RecsPresent being the number of destination queues in the list. It is valid for a distribution list to contain only one destination.

The value of RecsPresent must not be less than zero, and if it is greater than zero, ObjectType must be MQOT_Q; the call fails with reason code MQRC_RECS_PRESENT_ERROR if these conditions are not satisfied.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

ResponseRecOffset (MQLONG)
Offset of first response record from start of MQOD.

This is the offset in bytes fo the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. ResponseRecOffset is used only when a distribution list is being opened. The field is ignored if RecsPresent is zero.

When a distribution list is being opened, an array of one or more MQRR response records can be provided in order to identify the queues that failed to open (CompCode field in MQRR), and the reason for each failure (Reason field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed. That is, some queues were opened successfully while others failed, or failed for differing reasons. Reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the Reason parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied, there must be RecsPresent of them.

The response records can be provided in the same way as the object records; either by specifying an offset in ResponseRecOffset, or by specifying an address in ResponseRecPtr; see the description of ObjectRecOffset above for details of how to do this. However, no more than one of ResponseRecOffset and ResponseRecPtr can be used; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and reason code, from the put operation for a queue, replace those from the open operation for that queue only if the completion code from the latter was MQCC_OK or MQCC_WARNING.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

ResponseRecPtr (MQLONG)
Address of first response record.

This is the address of the first MQOR response record. ResponseRecPtr is used only when a distribution list is being opened. The field is ignored if RecsPresent is zero.

Either ResponseRecPtr or ResponseRecOffset can be used to specify the response records, but not both; see the description of the ResponseRecOffset field above for details. If ResponseRecPtr is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and all-null byte string otherwise. This field is ignored if Version is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer date type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQOD_STRUC_ID
Identifier for object descriptor structure.

For the C programming language, the constant MQOD_STRUC_ID_ARRAY is also defined; this has the same value as MQOD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQOD_STRUC_ID.

UnknownDestCount (MQLONG)
Number of remote queues opened successfully.

This is the number of queues in the distribution list that resolve to remote queues and that were opened successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if Version is less than MQOD_VERSION_2.

Version (MQLONG)
This is the structure version number; the value must be one of the following:

MQOD_VERSION_1
Version-1 object descriptor structure.

MQOD_VERSION_2
Version-2 object descriptor structure.

The following constant specifies the version number of the current version:

MQOD_CURRENT_VERSION
Current version of object descriptor structure. For WebSphere MQ for z/VSE, this field defaults to MQOD_VERSION_1.

The initial value of this field is MQOD_VERSION_1.

C declaration

```
typedef struct tagMQOD MQOD;
struct tagMQOD
{
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    ObjectType;       /* Object type */
  MQCHAR48  ObjectName;       /* Object name */
  MQCHAR48  ObjectQMgrName;   /* Object queue manager name */
  MQCHAR48  DynamicQName;     /* Dynamic queue name */
  MQCHAR12  AlternateUserId;  /* Alternate user identifier */
  MQLONG    RecsPresent;      /* Number of object records present */
  MQLONG    KnownDestCount;   /* Number of local queues opened
                                 successfully */
  MQLONG    UnknownDestCount; /* Number of remote queues opened
                                 successfully */
  MQLONG    InvalidDestCount; /* Number of queues that failed to
                                 open */
  MQLONG    ObjectRecOffset;  /* Offset of first object record from
                                 start of MQOD */
  MQLONG    ResponseRecOffset;/* Offset fo first response record from
                                 start of MQOD */
  MQPTR     ObjectRecPtr;     /* Address of first object record */
  MQPTR     ResponseRecPtr;   /* Address of first response record */
};MQOD;
typedef MQOD MQPOINTER PMQOD
```

COBOL declaration

```
  ** MQOD structure
   10 MQOD.
  ** Structure identifier
     15 MQOD-STRUCID PIC X(4) VALUE 'OD  '.
  ** Structure version number
     15 MQOD-VERSION PIC S9(9) BINARY VALUE 1.
  ** Object type
     15 MQOD-OBJECTTYPE PIC S9(9) BINARY VALUE 1.
  ** Object name
     15 MQOD-OBJECTNAME PIC X(48) VALUE SPACES.
  ** Object queue manager name
     15 MQOD-OBJECTQMGRNAME PIC X(48) VALUE SPACES.
  ** Dynamic queue name
     15 MQOD-DYNAMICQNAME PIC X(48) VALUE 'AMQ.*'.
  ** Alternate user identifier
     15 MQOD-ALTERNATEUSERID PIC X(12) VALUE SPACES.
  ** Number of object records present
     15 MQOD-RECSPRESENT PIC S9(9) BINARY VALUE 0.
  ** Number of local queues opened successfully
     15 MQOD-KNOWNDESTCOUNT PIC S9(9) BINARY VALUE 0.
  ** Number of remote queues opened successfully
     15 MQOD-UNKNOWNDESTCOUNT PIC S9(9) BINARY VALUE 0.
  ** Number of queues that failed to open
     15 MQOD-INVALIDDESTCOUNT PIC S9(9) BINARY VALUE 0.
  ** Offset of first object record from start of MQOD
     15 MQOD-OBJECTRECOFFSET PIC S9(9) BINARY VALUE 0.
  ** Offset of first response record from start of MQOD
     15 MQOD-RESPONSERECOFFSET PIC S9(9) BINARY VALUE 0.
  ** Address of first object record
     15 MQOD-OBJECTRECPTR POINTER VALUE NULL.
  ** Address of first response record
     15 MQOD-RESPONSERECPTR POINTER VALUE NULL.
```

PL/I declaration

```
dcl
 1 MQOD based,
  3 StrucId           char(4)
      init(MQOD_STRUC_ID),          /* Structure identifier */
  3 Version           fixed bin(31)
      init(MQOD_VERSION_1),         /* Structure version number */
  3 ObjectType        fixed bin(31)
      init(MQOT_Q),                 /* Object type */
  3 ObjectName        char(48)
      init(''),                     /* Object name */
  3 ObjectQMgrName    char(48)
      init(''),                     /* Object queue manager name */
  3 DynamicQName      char(48)
      init('AMQ.*')                 /* Dynamic queue name */
  3 AlternateUserId   char(12)
      init(''),                     /* Alternate user identifier */
  3 RecsPresent       fixed bin(31)
      init(0),                      /* Number of object records
                                       present */
  3 KnownDestCount    fixed bin(31)
      init(0),                      /* Number of local queues
                                       opened successfully */
  3 UnknownDestCount  fixed bin(31)
      init(0),                      /* Number of remote queues
                                       opened successfully */
  3 InvalidDestCount  fixed bin(31)
      init(0),                      /* Number of queues that failed
                                       to open */
  3 ObjectRecOffset   fixed bin(31)
      init(0),                      /* Offset of first object
                                       record from start of MQOD */
  3 ObjectRecPtr      pointer
      init(null()),                 /* Address of first object
                                       record */
  3 ResponseRecPtr    pointer
      init(null())                  /* Address of first response
                                       record */
```

MQOR - Object Record

The MQOR structure is used to specify the queue name and queue-manager name of a single destination queue. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

By providing and array of these structures on the MQOPEN call, it is possible to open a list of queues; this list is called a distribution list. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, provided that the queue was opened successfully.

Fields

Here is a summary of the fields.

Table 53. Fields in MQOR

Field	Description
ObjectName	Object name
ObjectQMgrName	Object Queue Manager Name

The MQOR structure contains the following fields; the fields are described in alphabetic order:

ObjectName (MQCHAR48)
This is the same as ObjectName field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.

- It must not be the name of a model queue.

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)
This is the same as the ObjectQMgrName field in the MQOD structure (see MQOD for details).

This is always an input field. The initial value of this field is the null string in C, and 48 bland characters in other programming languages.

C declaration

```
typedef struct tagMQOR {
struct tagMQOR
{
  MQCHAR48  ObjectName;       /* Object name */
  MQCHAR48  ObjectQMgrName;   /* Object queue manager name */
};MQOR;
typedef MQOR MQPOINTER PMQOR;
```

COBOL declaration

```
  ** MQOR structure
   10 MQOR.
  ** Object name
     15 MQOR-OBJECTNAME PIC X(48) VALUE SPACES.
  ** Object queue manager name
     15 MQOR-OBJECTQMGRNAME PIC X(48) VALUE SPACES.
```

PL/I declaration

```
  dcl
   1 MQOR based,
    3 Object name            char(48)
        init(''),                    /* Object name */
    3 ObjectQMgrName         char(48)
        init(''),                    /* Object queue manager name */
```

MQPD – Property descriptor

The MQPD is used to define the attributes of a property. The structure is an input/output parameter on the MQSETMP call and an output parameter on the MQINQMP call.

Data in MQPD must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 54. Fields in MQPD

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options controlling the action of MQPD

Table 54. Fields in MQPD (continued)

Field	Description
Support	Required support for message property
Context	Message context to which property belongs
CopyOptions	Copy options to which property belongs

Here is a description of the fields.

Context (MQLONG)
This describes what message context the property belongs to.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the Context field.

The following option can be specified:

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.

If the option previously described is not required, the following option can be used:

MQPD_NO_CONTEXT
The property is not associated with a message context.

An unrecognized value is rejected with a Reasoncode of MQRC_PD_ERROR.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQPD_NO_CONTEXT.

CopyOptions (MQLONG)
This describes which type of messages the property should be copied into. This is an output only field for recognized WebSphere MQ-defined properties. WebSphere MQ sets the appropriate value.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the CopyOptions field.

You can specify one or more of the options shown here. If you need more than one option, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

MQCOPY_FORWARD
This property is copied into a message being forwarded.

MQCOPY_REPLY
This property is copied into a reply message.

MQCOPY_REPORT
This property is copied into a report message.

MQCOPY_ALL
This property is copied into all types of subsequent messages.

The MQCOPY_DEFAULT option can be specified to supply the default set of copy options:

MQCOPY_DEFAULT
This property is copied into a message being forwarded, into a report message, or into a message received by a subscriber when a message is being published.

This is equivalent to specifying the combination of options MQCOPY_FORWARD, plus MQCOPY_REPORT, plus MQCOPY_PUBLISH.

If none of the options described above is required, use the MQCOPY_NONE option:

MQCOPY_NONE
Use this value to indicate that no other copy options have been specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties. This is an input/output field to the MQSETMP call and an output field from the MQINQMP call.

The initial value of this field is MQCOPY_DEFAULT.

Options (MQLONG)
The value must be:

MQPD_NONE
No options specified

This is always an input field. The initial value of this field is MQPD_NONE.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQPD_STRUC_ID
Identifier for property descriptor structure.

For the C programming language, the constant MQPD_STRUC_ID_ARRAY is also defined. This has the same value as MQPD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQPD_STRUC_ID.

Support (MQLONG)
This field describes what level of support for the message property is required of the queue manager, in order for the message containing this property to be put to a queue. This applies only to WebSphere MQ-defined properties; support for all other properties is optional.

The field is automatically set to the correct value when the WebSphere MQ-defined property is known by the queue manager. If the property is not recognized, MQPD_SUPPORT_OPTIONAL is assigned. When a queue manager receives a message containing a WebSphere MQ- defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the Support field.

When setting a WebSphere MQ-defined property using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option was set, Support becomes an input field. This allows an application to put a WebSphere MQ-defined property with the correct value, where the property is unsupported by the connected queue manager, but where the message is intended to be processed on another queue manager.

The value MQPD_SUPPORT_OPTIONAL is always assigned to properties that are not WebSphere MQ-defined properties.

One of the following values is returned by the MQINQMP call, or one of the values can be specified when using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option is set:

MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported.

The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not WebSphere MQ-defined.

MQPD_SUPPORT_REQUIRED
Support for the property is required. The message is rejected by a queue manager that does not support the WebSphere MQ-defined property. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

MQPD_SUPPORT_REQUIRED_IF_LOCAL
The message is rejected by a queue manager that does not support the WebSphere MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

This is an output field on the MQINQMP call and an input field on the MQSETMP call if the message handle was created with the MQCMHO_NO_VALIDATION option set. The initial value of this field is MQPD_SUPPORT_OPTIONAL.

Version (MQLONG)
This is the structure version number. The value must be:

MQPD_VERSION_1
Version-1 property descriptor structure.

This constant specifies the version number of the current version:

MQPD_CURRENT_VERSION
Current version of property descriptor structure.

This is always an input field. The initial value of this field is MQPD_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQPD.

Table 55. Initial values of fields in MQPD

Field name	Name of constant	Value of constant
StrucId	MQPD_STRUC_ID	'PD'
Version	MQPD_VERSION_1	1
Options	MQPD_NONE	0
Support	MQPD_SUPPORT_OPTIONAL	0
Context	MQPD_NO_CONTEXT	0
CopyOptions	MQCOPY_DEFAULT	0

Note: In the C programming language, the macro variable MQPD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQPD MyPD = {MQPD_DEFAULT};
```

C declaration

```
struct tagMQPD {
  MQCHAR4  StrucId;     /* Structure identifier */
  MQLONG   Version;     /* Structure version number */
  MQLONG   Options;     /* Options that control the action of MQSETMP
                           and MQINQMP */
  MQLONG   Support;     /* Property support option */
  MQLONG   Context;     /* Property context */
  MQLONG   CopyOptions; /* Property copy options */
};
```

COBOL declaration

```
** MQPD structure
 10  MQPD.
** Structure identifier
 15  MQPD-STRUCID                      PIC X(4).
** Structure version number
 15  MQPD-VERSION                      PIC S9(9) BINARY.
** Options that control the action of MQSETMP and MQINQMP
 15  MQPD-OPTIONS                      PIC S9(9) BINARY.
** Property support option
 15  MQPD-SUPPORT                      PIC S9(9) BINARY.
** Property context
 15  MQPD-CONTEXT                      PIC S9(9) BINARY.
** Property copy options
 15  MQPD-COPYOPTIONS                  PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQPD based,
  3 StrucId          char(4)
    init(MQPD_STRUC_ID),         /* Structure identifier */
  3 Version          fixed bin(31)
    init(MQPD_VERSION_1),        /* Structure version number */
  3 Options          fixed bin(31)
    init(MQPD_NONE),             /* Options that control the action */
                                 /* of MQSETMP and MQINQMP */
  3 Support          fixed bin(31)
    init(MQPD_SUPPORT_OPTIONAL), /* Property support option */
  3 Context          fixed bin(31)
    init(MQPD_NO_CONTEXT),       /* Property context */
  3 CopyOptions      fixed bin(31)
    init(MQCOPY_DEFAULT);        /* Property copy options */
```

MQPMO – Put message options

The MQPMO structure allows the application to specify options that control how messages are placed on queues. The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Fields

Here is a summary of the fields.

Table 56. Fields in MQPMO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options that control the action of MQPUT and MQPUT1
Timeout	Reserved
Context	Object handle of input queue
KnownDestCount	Number of messages sent successfully to local queues
UnknownDestCount	Number of messages sent successfully to remote queues
InvalidDestCount	Number of messages that could not be sent
ResolvedQName	Resolved name of destination queue
ResolvedQMgrName	Name Resolved name of destination queue manager
Note: The remaining fields are ignored if Version is less than MQPMO_VERSION_2.	
RecsPresent	Number of put message records or response records present
PutMsgRecFields	Flags indicating which MQPMR fields are present
PutMsgRecOffset	Offset of first put message record from start of MQPMO
ResponseRecOffset	Offset of first response record from start of MQPMO
PutMsgRecPtr	Address of first put message record
ResponseRecPtr	Address of first response record

Table 56. Fields in MQPMO (continued)

Field	Description
Note: The remaining fields are ignored if Version is less than MQPMO_VERSION_3.	
OriginalMsgHandle	Original message handle
NewMsgHandle	New message handle
Action	New message handle NewMsgHandle Action Type of put being performed and the relationship between the original message specified by the OriginalMsgHandle field and the new message specified by the NewMsgHandle field
PubLevel	Level of subscription targeted by the publication

Here is a description of the fields.

Action (MQLONG)

This specifies the type of put being performed and the relationship between the original message specified by the OriginalMsgHandle field and the new message specified by the NewMsgHandle field. The properties of the message are chosen by the queue manager according to the value of the Action specified.

If the MsgDesc parameter is not supplied, then the message descriptor for the new message is populated from the message handle fields of the MQPMO, according to the rules described in this topic.

If an incorrect action value is specified, the call fails with the reason code MQRC_ACTION_ERROR.

Any one of the following actions can be specified:

MQACTP_NEW

A new message is being put, and no relationship to a previous message is being specified by the program. The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, this is used as the message descriptor unmodified
- If a MsgDesc is not supplied, then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.

Message data is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_FORWARD

A previously retrieved message is being forwarded. The original message handle specifies the message that was previously retrieved. The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.
- If MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified in the MQPMO.Options, then these are honoured.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_FORWARD in the MQPD.CopyOptions.
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_REPLY

A reply is being made to a previously retrieved message. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call,, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied then initial message descriptor fields are chosen as shown in Table 57.

Table 57. Reply message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set, MQRO_DISCARD_MSG; otherwise, MQRO_NONE
MsgType	MQMT_REPLY
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set, copied from the input message; otherwise, MQEI_UNLIMITED
Feedback	MQFB_NONE
MsgId	If MQPMO_NEW_MSG_ID is set, a new message identifier is generated; else, if MQRO_PASS_MSG_ID is set, copied from the input message; otherwise, MQMI_NONE

Table 57. Reply message handle transformation (continued)

Field in MQMD	Value used
CorrelId	If MQPMO_NEW_CORREL_ID is set, a new correlation identifier is generated; else, if MQRO_COPY_MSG_ID_TO_CORREL_ID is set, copied from the MsgId field of the input message; else, if MQRO_PASS_CORREL_ID is set, copied from the CorrelId field of the input message; otherwise, MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- The message descriptor is then modified by the new message handle. Any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPLY in the MQPD.CopyOptions.
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT/MQPUT1 Buffer parameter.

MQACTP_REPORT

A report is being generated as a result of a previously retrieved message. The original message handle specifies the message causing the report to be generated.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options, then initial message descriptor fields are chosen as shown in Table 58 on page 830.

Table 58. Report message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set, MQRO_DISCARD_MSG; otherwise, MQRO_NONE
MsgType	MQMT_REPORT
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set, copied from the input message; otherwise, MQEI_UNLIMITED
MsgId	If MQPMO_NEW_MSG_ID is set, a new message identifier is generated; else, if MQRO_PASS_MSG_ID is set, copied from the input message; otherwise, MQMI_NONE
CorrelId	If MQPMO_NEW_CORREL_ID is set, a new correlation identifier is generated; else, if MQRO_COPY_MSG_ID_TO_CORREL_ID is set, copied from the MsgId field of the input message; else, if MQRO_PASS_CORREL_ID is set, copied from the CorrelId field of the input message; otherwise, MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
OriginalLength	Set to the BufferLength

- The message descriptor is then modified by the new message handle. Any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPORT in the MQPD.CopyOptions.
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case, the property is removed from the message.

The Feedback field in the resultant MQMD represents the report that is to be generated. A Feedback value of MQFB_NONE causes the MQPUT or MQPUT1 call to fail with reason code MQRC_FEEDBACK_ERROR.

To choose the user data of the report message, WebSphere MQ consults the Report and Feedback fields in the resultant MQMD, and the Buffer and BufferLength parameters of the MQPUT or MQPUT1 call.

- If Feedback is MQFB_COA, MQFB_COD, or MQFB_EXPIRATION, then the value of Report is inspected.
- If any of the following cases is true, the full message data from Buffer for a length of BufferLength is used.
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_FULL_DATA.

  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_FULL_DATA.
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_FULL_DATA.
- If any of the following cases is true, the first 100 bytes of the message (or BufferLength if this is less than 100) from Buffer are used
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_DATA.
  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_DATA.
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_DATA.
- If Feedback is MQFB_EXPIRATION, MQFB_COD, or MQFB_COA, and Report does not contain the *_WITH_FULL_DATA or *_WITH_DATA options relevant to that Feedback value, then no user data is included with the message.
- If Feedback takes a different value from those listed above, then Buffer and BufferLength are used as normal.

The derivation of the user data is shown in Table 59.

Table 59. Source of user data

	MQFB_COA	**MQFB_COD**	**MQFB_EXPIRATION**
MQRO_EXPIRATION_WITH_FULL_DATA	None	None	Buffer(Bufferlength)
MQRO_COD_WITH_FULL_DATA	None	Buffer(Bufferlength)	None
MQRO_COA_WITH_FULL_DATA	Buffer(Bufferlength)	None	None
MQRO_EXPIRATION_WITH_DATA	None	None	Buffer(First 100 bytes)
MQRO_COD_WITH_DATA	None	Buffer(First 100 bytes)	None
MQRO_COA_WITH_DATA	Buffer(First 100 bytes)	None	None

Context (MQHOBJ)
If MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If neither MQPMO_PASS_IDENTITY_CONTEXT nor MQPMO_PASS_ALL_CONTEXT is specified, this field is ignored.

This field is not supported by WebSphere MQ for z/VSE.

InvalidDestCount (MQLONG)
This is the number of messages that could not be sent to queues in the distribution list. The count includes queues that failed to open, as well as queues that were opened successfully but for which the put operation failed.

This field is also set when putting a message to a single queue that is not in a distribution list.

This field is not supported by WebSphere MQ for z/VSE.

KnownDestCount (MQLONG)
This is the number of messages that the current MQPUT or MQPUT1 call

has sent successfully to queues in the distribution list that are local queues. The count does not include messages sent to queues that resolve to remote queues (even though a local transmission queue is used initially to store the message).

This field is also set when putting a message to a single queue that is not in a distribution list.

This field is not supported by WebSphere MQ for z/VSE.

NewMsgHandle (MQHMSG)
This is an optional handle to the message being put, subject to the value of the Action field. It defines the properties of the message and overrides the values of the OriginalMsgHandle, if specified.

On return from the MQPUT or MQPUT1 call, the contents of the handle reflect the message that was actually put.

This is an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQPMO_VERSION_3.

Options (MQLONG)
Any or none of the following can be specified. If more than one is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; any other combinations are valid.

The following *syncpoint* options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

MQPMO_SYNCPOINT
The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If neither this option nor MQPMO_NO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment. On z/VSE, the message is included in the current unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_SYNCPOINT with MQPMO_NO_SYNCPOINT.

MQPMO_NO_SYNCPOINT
The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If neither this option nor MQPMO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment. On z/VSE, the message is included in the current unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

The following *Message-identifier and correlation-identifier* options request the queue manager to generate a new message identifier or correlation identifier:

MQPMO_NEW_MSG_ID
The queue manager replaces the contents of the MsgId field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

Using this option relieves the application of the need to reset the MsgId field to MQMI_NONE prior to each MQPUT or MQPUT1 call.

MQPMO_NEW_CORREL_ID
The queue manager replaces the contents of the CorrelId field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

MQPMO_NEW_CORREL_ID is useful in situations where the application requires a unique correlation identifier.

The following *group and segment* option relates to the processing of messages in groups and segments of logical messages. See "Message grouping and segmentation" on page 959 for definitions of these terms.

MQPMO_LOGICAL_ORDER
This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call.

If MQPMO_LOGICAL_ORDER is specified, it indicates that the application will use successive MQPUT calls to:

- Put the segments in each logical message in the order of increasing segment offset, starting from 0, with no gaps.
- Put all the segments in one logical message before putting the segments in the next logical message.
- Put the logical messages in each message group in the order of increasing message sequence number, starting from 1, with no gaps.
- Put all the logical messages in one message group before putting logical messages in the next message group.

The above order is called logical order.

Because the application has told the queue manager how it will put messages in groups and segments of logical messages, the application does not have to maintain and update the group and segment information on each MQPUT call, because the queue manager does this. Specifically, it means that the application does not need to set the GroupId, MsgSeqNumber, and Offset fields in

MQMD, because the queue manager sets these to the appropriate values. The application needs to set only the MsgFlags field in MQMD, to indicate when messages belong to groups or are segments of logical messages, and to indicate the last message in a group or last segment of a logical message.

Once a message group or logical message has been started, subsequent MQPUT calls must specify the appropriate MQMF_* flags in MsgFlags in MQMD. If the application tries to put a message that is not in a group when there is an unterminated message group, or put a message that is not a segment when there is an unterminated logical message, the call fails with reason code MQRC_INCOMPLETE_GROUP or MQRC_INCOMPLETE_MSG, as appropriate. However, the queue manager retains the information about the current message group or current logical message, and the application can terminate them by sending a message (possibly with no application message data) specifying MQMF_LAST_MSG_IN_GROUP or MQMF_LAST_SEGMENT as appropriate, before reissuing the MQPUT call to put the message that is not in the group or not a segment.

When MQPMO_LOGICAL_ORDER is specified, the MQMD supplied on the MQPUT call must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_MD_VERSION.

If MQPMO_LOGICAL_ORDER is not specified, messages in groups and segments of logical messages can be put in any order, and it is not necessary to put complete message groups or complete logical messages. It is the application's responsibility to ensure that the GroupId, MsgSeqNumber, Offset, and MsgFlags fields have appropriate values.

Use the following technique to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the GroupId, MsgSeqNumber, Offset, MsgFlags, and Persistence fields to the appropriate values, and then issue the MQPUT call with MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT set as desired, but without specifying MQPMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQPUT calls using that queue handle can specify MQPMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQPUT call is separate from the group and segment information that it retains for the MQGET call.

For any given queue handle, the application can mix MQPUT calls that specify MQPMO_LOGICAL_ORDER with MQPUT calls that do not, but note the following points:

- If MQPMO_LOGICAL_ORDER is not specified, each successful MQPUT call causes the queue manager to set the group and segment information for the queue handle to the values specified by the application; this replaces the existing group and segment information retained by the queue manager for the queue handle.

- If MQPMO_LOGICAL_ORDER is not specified, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
  MQRC_INCOMPLETE_GROUP
  MQRC_INCOMPLETE_MSG

Note: The queue manager does not check the group and segment information for the MQPUT1 call.

For applications that put messages and segments in logical order, specify MQPMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQPMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the GroupId, MsgSeqNumber, Offset, and MsgFlags fields in MQMD are set correctly, before each MQPUT or MQPUT1 call.

For example, an application that wants to forward physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, must not specify MQPMO_LOGICAL_ORDER. There are two reasons for this:

- If the messages are retrieved and put in order, specifying MQPMO_LOGICAL_ORDER assigns a new group identifier to the messages, and this might make it difficult or impossible for the originator of the messages to correlate any reply or report messages that result from the message group.
- In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By specifying neither MQPMO_LOGICAL_ORDER nor the corresponding MQGMO_LOGICAL_ORDER on the MQGET call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

Applications that generate report messages for messages in groups or segments of logical messages must also not specify MQPMO_LOGICAL_ORDER when putting the report message.

MQPMO_LOGICAL_ORDER can be specified with any of the other MQPMO_* options.

The following options affect what happens when the queue manager is quiescing:

MQPMO_FAIL_IF_QUIESCING
This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

The call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING.

If you need none of the options described, use the following default option:

MQPMO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values.

MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the Options field is MQPMO_NONE.

OriginalMsgHandle (MQHMSG)
This is an optional handle to a message. It may have been previously retrieved from a queue. The use of this handle is subject to the value of the Action field. See also NewMsgHandle.

The contents of the original message handle are not altered by the MQPUT or MQPUT1 call.

This is an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQPMO_VERSION_3.

PubLevel (MQLONG)
This is a reserved field in WebSphere MQ for z/VSE.

The initial value of this field is 9.

PutMsgRecFields (MQLONG)
Flags indicating which MQPMR fields are present.

This field contains flags that must be set to indicate which MQPMR fields are present in the put message records provided by the application. PutMsgRecFields is used only when the message is being put to a distribution list. The field is ignored if RecsPresent is zero, or both PutMsgRecOffset and PutMsgRecPtr are zero.

For fields that are present, the queue manager uses, for each destination, the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

One or more of the following flags can be specified to indicate which fields are present in the put message records:

- MQPMRF_MSF_ID Message-identifier field is present.
- MQPMRF_CORREL_ID Correlation-identifier field is present.
- MQPMRF_GROUP_ID Group-identifier field is present.
- MQPMRF_FEEDBACK Feedback field is present.
- MQPMRF_ACCOUNTING_TOKEN Accounting-token field is present.

If this flag is specified, either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT must be specified in the Options field. If this condition is not satisfied, the call fails with reason code MQ_PMO_RECORD_FLAGS_ERROR.

If no MQPMR fields are present, the following can be specified:

- MQPMRF_NONE No put-message record fields are present.

If this value is specified, either RecsPresent must be zero, or both PutMsgRecOffset and PutMsgRecPtr must be zero.

MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

If PutMsgRecFields contains flags which are not valid, or put message records are provided but PutMsgRecFields has the value MQPMRF_NONE, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

This is an input field. The initial value of this field is MQPMRF-NONE. This field is ignored if Version is less than MQPMO_VERSION_2.

PutMsgRecOffset (MQLONG)

Offset fo first put message record from start of MQPMO.

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. PutMsgRecOffset is used only when the message is being put to a distribution list. The field is ignored if RecsPresent is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually. These properties are:

- Message identifier
- Correlation identifier
- Group identifier
- Feedback value
- Accounting token

It is not necessary to specify all of these properties, but whatever subset is chosen, the fields must be specified in the correct order. See the description of the MQPMR structure for further details.

Usually, there should be as many put messages as there are object records specified by the MQOD when the distribution list is opened. Each put message record supplies the message properties for the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have message records allocated for them at the appropriate positions in the array, although the message properties are ignored in this case.

It is possible for the number of put message records to differ from the number of object records. If there are fewer put message records than object records, the message properties for the destinations, which do not have put message records, are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excesses are not used (although it must still be possible to access them). Put message records are optional, buf if they are supplied, there must be RecsPresent of them

The put message records can be provided in a similar way to the object records in MQOD, either by specifying an offset in PutMsgRecOffset, or by specifying an address in PutMsgRecPtr. For details of how to do this, see "MQOD - Object descriptor" above.

No more than one of PutMsgRecOffset and PutMsgRecPtr can be used; the call fails with reason code MQRC_PUT_MSG_RECORDS_ERROR if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQPMO_VERSION_2.

PutMsgRecPtr (MQPTR)
Address of first put message record.

This is the address of the first MQPMR put message record. PutMsgRecPtr is used only when the message is being put to a distribution list. The field is ignored if RecsPresent is zero.

Either PutMsgRecPtr of PutMsgRecOffset can be used to specify the put message records, but not both. See the description of the PutMsgRecOffset field above for details. If PutMsgRecPtr is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if Version is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

RecsPresent (MQLONG)
Number of put message records or response records present.

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional. The application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide RecsPresent records of each type.

The value of RecsPresent need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess is not used. If too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see PutMsgRecOffset below).

If RecsPresent is less than zero, or is greater than zero, but the message is not being put to a distribution list, the call fails with reason code MQRC_RECS_PRESENT_ERROR.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQPMO_VERSION_2.

ResolvedQMgrName (MQCHAR48)
This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by ResolvedQName, and can be the name of the local queue manager.

A nonblank value is returned if the object is a single queue.

This is an output field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResolvedQName (MQCHAR48)
This is the name of the destination queue after name resolution has been

performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by ResolvedQMgrName.

A nonblank value is returned if the object is a single queue.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResponseRecOffset (MQLONG)
Offset of first response record from start of MQPMO.

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. ResponseRecOffset is used only when the message is being put to a distribution list. The field is ignored if RecsPresent is zero.

When the message is being put to a distribution list, an array of one or more MQRR response records can be provided in order to identify the queues to which the message was not sent successfully (CompCode field in MQRR), and the reason for each failure (Reason field in MQRR). The message may not have been sent, either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is mixed, that is some messages were sent successfully, while others failed, or all failed for differing reasons. Reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the Reason parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there should be as many response records as there are object records specified by MQOD when the distribution list is opened. When necessary, each response record is set to the completion code and reason code for the put to the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have response records allocated for them at the appropriate positions in the array, although they are set to the complication code and reason code resulting from the open operation, rather than the put operation.

It is possible for the number of response records to differ from the number of object records. If there are fewer response records than object records, it may not be possible for the application to identify all of the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records, the excesses are not used, although it must still be possible to access them. Response records are optional, but if they are supplied, there must be RecsPresent of them.

The response records can be provided in a similar way to the object records in MQOD, either by specifying an offset in ResponseRecOffset, or by specifying an address in ResponseRecPtr. For details on how to do this, see "MQOD - object descriptor" above.However, no more than one of ResponseRecOffset and ResponseRecPtr can be used. The call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if Version is less than MQPMO_VERSION_2.

ResponseRecPtr (MQPTR)
Address of first response record.

This is the address of the first MQRR response record. ResponseRecPtr is used only when the message is being put to a distribution list. The field is ignored if RecsPresent is zero.

Either ResponseRecPtr or ResponseRecOffset can be used to specify the response records, but not both. See the description of the ResponseRecOffset field above for details. If ResponseRecPtr is not used, it must be set to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information, if requested, is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if Version is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ResolvedQName (MQCHAR48)
This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by ResolvedQMgrName.

A nonblank value is returned if the object is a single queue.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQPMO_STRUC_ID
Identifier for put-message options structure.

For the C programming language, the constant MQPMO_STRUC_ID_ARRAY is also defined; this has the same value as MQPMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQPMO_STRUC_ID.

Timeout (MQLONG)
This is a reserved field; its value is not significant. The initial value of this field is ?1.

UnknownDestCount (MQLONG)
This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that resolve to remote queues. Messages that the queue manager retains temporarily in distribution-list form count as the number of individual destinations that those distribution lists contain.

This field is not supported by WebSphere MQ for z/VSE.

Version (MQLONG)
This is the structure version number; the value must be one of the following:

MQPMO_VERSION_1
Version-1 put-message options structure.

MQPMO_VERSION_2
Version-2 put-message options structure.

The following constant specifies the version number of the current version:

MQPMO_CURRENT_VERSION
Current version of put-message options structure. For WebSphere MQ for z/VSE, this field defaults to MQPMO_VERSION_1.

The initial value of this field is MQPMO_VERSION_1.

C declaration

```
typedef struct tagMQPMO {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    Options;          /* Options that control the action of MQPUT or MQPUT1 */
  MQLONG    Timeout;          /* Reserved */
  MQHOBJ    Context;          /* Object handle of input queue */
  MQLONG    KnownDestCount;   /* Reserved */
  MQLONG    UnknownDestCount; /* Reserved */
  MQLONG    InvalidDestCount; /* Reserved */
  MQCHAR48  ResolvedQName;    /* Resolved name of destination queue */
  MQCHAR48  ResolvedQMgrName; /* Resolved name of destination queue manager */
  MQLONG    RecsPresent;      /* Number of put message records or response records
                                 present */
  MQLONG    PutMsgRecFields;  /* Flags indicating which MQPMR fields are present */
  MQLONG    PutMsgRecOffset;  /* Offset of first put message record from start
                                 of MQPMO */
  MQLONG    ResponseRecOffset;/* Offset of first response record from start of MQPMO */
  MQPTR     PutMsgRecPtr;     /* Address of first put message record */
  MQPTR     ResponseRecPtr;   /* Address of first response record */
} MQPMO;
typedef MQPMO MQPOINTER PMQPMO;
```

COBOL declaration

```
** MQPMO structure
 10 MQPMO.
** Structure identifier
  15 MQPMO-STRUCID PIC X(4) VALUE 'PMO'.
** Structure version number
  15 MQPMO-VERSION PIC S9(9) BINARY VALUE 1.
** Options that control the action of MQPUT and MQPUT1
  15 MQPMO-OPTIONS PIC S9(9) BINARY VALUE 0.
** Reserved
  15 MQPMO-TIMEOUT PIC S9(9) BINARY VALUE -1.
** Object handle of input queue
  15 MQPMO-CONTEXT PIC S9(9) BINARY VALUE 0.
** Number of messages sent successfully to local queues
  15 MQPMO-KNOWNDESTCOUNT PIC S9(9) BINARY VALUE 0.
** Number of messages sent successfully to remote queues
  15 MQPMO-UNKNOWNDESTCOUNT PIC S9(9) BINARY VALUE 0.
** Number of messages that could not be sent
  15 MQPMO-INVALIDDESTCOUNT PIC S9(9) BINARY VALUE 0.
** Resolved name of destination queue
  15 MQPMO-RESOLVEDQNAME PIC X(48) VALUE SPACES.
** Resolved name of destination queue manager
  15 MQPMO-RESOLVEDQMGRNAME PIC X(48) VALUE SPACES.
** Number of put message records or response records present
  15 MQPMO-RECSPRESENT PIC S9(9) BINARY VALUE 0.
** Flags indicating which MQPMR fields are present
  15 MQPMO-PUTMSGRECFIELDS PIC S9(9) BINARY VALUE 0.
** Offset of first put message record from start of MQPMO
  15 MQPMO-PUTMSGRECOFFSET PIC S9(9) BINARY VALUE 0.
** Offset of first response record from start of MQPMO
  15 MQPMO-RESPONSERECOFFSET PIC S9(9) BINARY VALUE 0.
** Address of first put message record
  15 MQPMO-PUTMSGRECPTR POINTER VALUE NULL.
** Address of first response record
  15 MQPMO-RESPONSERECPTR POINTER VALUE NULL.
```

PL/I declaration

```
dcl
 1 MQPMO based,
  3 StrucId            char(4)
      init(MQPMO_STRUC_ID),         /* Structure identifier */
  3 Version            fixed bin(31)
      init(MQPMO_VERSION_1),        /* Structure version number */
  3 Options            fixed bin(31)
      init(MQPMO_NONE),             /* Options that control the action of MQPUT and MQPUT1 */
  3 Timeout            fixed bin(31)
      init(-1),                     /* Reserved */
  3 Context            fixed bin(31)
      init(0),                      /* Object handle of input queue */
  3 KnownDestCount     fixed bin(31)
      init(0),                      /* Reserved */
  3 UnknownDestCount   fixed bin(31)
      init(0),                      /* Reserved */
  3 InvalidDestCount   fixed bin(31)
      init(0),                      /* Reserved */
  3 ResolvedQName      char(48)
      init(''),                     /* Resolved name of destination queue */
  3 ResolvedQMgrName   char(48)
      init(''),                     /* Resolved name of destination queue manager */
  3 RecsPresent        fixed bin(31)
      init(0),                      /* Number of put message records or response
                                       records present */
  3 PutMsgRecFields    fixed bin(31)
      init(MQPMRF_NONE),            /* Flags indicating which MQPMR fields are
                                       present */
  3 PutMsgRecOffset    fixed bin(31)
      init(0),                      /* Offset of first put message record from
                                       start of MQPMO */
  3 ResponseRecOffset  fixed bin(31)
      init(0)                       /* Offset of first response record from start
                                       of MQPMO */
  3 PutMsgRecPtr       pointer
      init(null()),                 /* Address of first put message record */
  3 ResponseRecPtr     pointer
      init(null()),                 /* Address of first response */
```

MQPMR – Put message record

The MQPMR structure is used to specify various message properties for a single destination when a message is being put to a distribution list. MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

By providing an array of these structures on the MQPUT or MQPUT1 call, it is possible to specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/ output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated bay the flags in the PutMsgRecFields field in MQPMO. Fields that are present must occur in the following order:

- MsgId
- CorrelId
- GroupId
- Feedback
- AccountingToken

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the header, COPY, and INCLUDE files for the supported programming languages. The application programmer should create a declaration containing the fields that are required by the application, and set the flags in PutMsgRecFields to indicate the fields that are present.

```
typedef struct tagMQPMR MQPMR;
struct tagMQPMR
{
  MQBYTE24  MsgId;
  MQBYTE24  CorrelId;
} MQPMRa;
```

or another example:

```
typedef struct tagMQPMRb MQPMRb;
struct tagMQPMRb
{
  MQBYTE24  CorrelId;
  MQBYTE24  GroupId;
  MQBYTE24  Feedback;
};MQPMRb;
```

MQRFH2 - Rules and formatting header 2

The MQRFH2 header is based on the MQRFH header, but it allows Unicode strings to be transported without translation, and it can carry numeric datatypes.

The MQRFH2 structure defines the format of the version-2 rules and formatting header. Use this header to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name

MQFMT_RF_HEADER_2

Character set and encoding

Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than NameValueData are in the character set and encoding given by the CodedCharSetId and Encoding fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

  The character set must be one that has single-byte characters for the characters that are valid in queue names.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts these fields to the requested character set and encoding.
- NameValueData is in the character set given by the NameValueCCSID field. Only certain Unicode character sets are valid for NameValueCCSID (see the description of NameValueCCSID for details).

  Some character sets have a representation that depends on the encoding. If NameValueCCSID is one of these character sets, NameValueData must be in the same encoding as the other fields in the MQRFH2.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts NameValueData to the requested encoding, but does not change its character set.

Fields

Here is a summary of the fields.

Table 60. Fields in MQRFH2

Field	Description
StrucId	Structure identifier
Version	Structure version number
StrucLength	Length in bytes of the MQRFH2 structure
Encoding	Numeric encoding of the data that follows the last NameValueData field
CodedCharSetId	CCSID of the data that follows the last NameValueData field.
Format	Format name of the data that follows the last NameValueData field.
Flags	Flags
NameValueCCSID	CCSID of the data in the NameValueData field.

Here is a description of the fields.

CodedCharSetId (MQLONG)

This specifies the character set identifier of the data that follows the last NameValueData field. It does not apply to character data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. This special value can be used:

MQCCSI_INHERIT
Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call. MQCCSI_INHERIT cannot be used if the value of the PutApplType field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG)
This specifies the numeric encoding of the data that follows the last NameValueData field. It does not apply to numeric data in the MQRFH2 structure itself. On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)
This value must be specified:

MQRFH_NONE
No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8)
This specifies the format name of the data that follows the last NameValueData field.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the Format field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueCCSID (MQLONG)
This specifies the coded character set identifier of the data in the NameValueData field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last NameValueData field at the end of the structure.

NameValueCCSID must have one of the values shown in Table 61.

Table 61. NameValueCCSID (MQLONG) valid values

CCSID	Meaning
1200	UCS-2 open-ended
13488	UCS-2 2.0 subset
17584	UCS-2 2.1 subset (includes the Euro symbol)
1208	UTF-8

For the UCS-2 character sets, the encoding (byte order) of the NameValueData must be the same as the encoding of the other fields in the MQRFH2 structure. Surrogate characters (X'D800' through X'DFFF') are not supported.

Note: If NameValueCCSID does not have one of the values listed in Table 61 on page 845, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code MQRC_SOURCE_CCSID_ERROR and the message is returned unconverted.

The initial value of this field is 1208.

NameValueData (MQCHARn)

This is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of this string is given by the NameValueLength field that precedes the NameValueData field. This length must be a multiple of four.

The NameValueLength and NameValueData fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required. For example: *length1 data1 length2 data2 length3 data3*.

Note:

1. Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.
2. For further information on the method of terminating the following NameValue fields, see topic NameValueString.

NameValueData is not converted to the character set specified on the MQGET call when the message is retrieved with the MQGMO_CONVERT option in effect; NameValueData remains in its original character set. However, NameValueData is converted to the encoding specified on the MQGET call.

Syntax of name/value data:

The string consists of a single folder that contains zero or more properties. The folder is delimited by XML start and end tags whose name is the name of the folder:

<folder> property1 property2 ... </folder>

Optionally, the content='properties' element can be included in the folder start tag. This indicates that the content of the folder is to be treated as message properties. This element must only be used with user-defined folders and not IBM-defined folders. For example, <wmq> or <jms>.

For example:

```
<com.ourcompany content='properties'> ... </com.ourcompany>
```

Characters following the folder end tag, up to the length defined by NameValueLength, must be blank. Within the folder, each property is composed of a name and a value, and optionally a data type:

```
<name dt="datatype">value</name>
```

In these examples:

- Specify the delimiter characters (<, =, ?, /, and >) exactly as shown. The parameter *value* can be wrapped in either double quotes or apostrophes, for example dt="*datatype*" or dt='*datatype*'
- *name* is the user-specified name of the property. See below for more information about names.

- *datatype* is an optional user-specified data type of the property. See below for valid data types.
- *value* is the user-specified value of the property. See below for more information about values.
- Blanks are significant between the > character that precedes a value, and the < character that follows the value, and at least one blank must precede dt=. Elsewhere you can code blanks freely between tags, or preceding or following tags (for example, in order to improve readability); these blanks are not significant.

If properties are related to each other, you can group them together by enclosing them within XML start and end tags whose name is the name of the group:

```
<folder> <group> property1 property2 ... </group> </folder>
```

Groups can be nested within other groups, without limit, and a given group can occur more than once within a folder. A folder can also contain some properties in groups and other properties not in groups.

Names of properties, groups, and folders:

The names of properties, groups, and folders must be valid XML tag names, with the exception of the colon character (:), which is not permitted in a property, group, or folder name. In particular:

- Names must start with a letter or an underscore. Valid letters are defined in the W3C XML specification and consist essentially of Unicode categories Ll, Lu, Lo, Lt, and Nl.
- The remaining characters in a name can be letters, decimal digits, underscores, hyphens, or dots. These correspond to Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd.
- The Unicode compatibility characters (X'F900' and above) are not permitted in any part of a name.
- Names must not start with the string XML in any mixture of uppercase or lowercase. In addition:
  - Names are case-sensitive. For example, ABC, abc, and Abc are treated as three different names.
  - Each folder has a separate name space. As a result, a group or property in one folder does not conflict with a group or property of the same name in another folder.
  - Groups and properties occupy the same name space within a folder. As a result, a property cannot have the same name as a group within the folder containing that property.

Generally, programs that analyze the NameValueData field must ignore properties or groups that have names that the program does not recognize, provided that those properties or groups are correctly formed.

Data types of properties:

Each property can have an optional data type. If specified, the data type must be one of the values shown in Table 62 on page 848, in uppercase, lowercase, or mixed case.

Table 62. Data types of properties

Data type	Used for
string	Any sequence of characters. Certain characters must be specified using escape sequences (see below).
boolean	The character 0 or 1 (1 denotes TRUE).
bin.hex	Hexadecimal digits representing octets.
i1	Integer number in the range -128 through +127, expressed using only decimal digits and optional sign.
i2®	Integer number in the range -32 768 through +32 767, expressed using only decimal digits and optional sign.
i4	Integer number in the range -2 147 483 648 through +2 147 483 647, expressed using only decimal digits and optional sign.
int	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign. This can be used in place of i1, i2, i4, or i8 if the sender does not wish to imply a particular precision.
r4	Floating-point number with magnitude in the range 1.175E-37 through 3.402 823 47E+38, expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.
r8	Floating-point number with magnitude in the range 2.225E-307 through 1.797 693 134 862 3E+308 expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.

Values of properties:

The value of a property can consist of any characters, except as shown in Table 63. Each occurrence in the value of a character marked as mandatory must be replaced by the corresponding escape sequence. Each occurrence in the value of a character marked as optional can be replaced by the corresponding escape sequence, but this is not required.

Table 63. Data types of properties

Character	Escape sequence	Usage
&	&amp;	Mandatory
<	<	Mandatory
>	>	Optional
"	"	Optional
'	'	Optional

Note: The & character at the start of an escape sequence must not be replaced by &.

In the example shown here, the blanks in the value are significant. However, no escape sequences are needed:

```
<Famous_Words>The program displayed "Hello World"</Famous_Words>
```

NameValueLength (MQLONG)

This specifies the length in bytes of the data in the NameValueData field. To avoid problems with data conversion of the data (if any) that follows the NameValueData field, NameValueLength must be a multiple of four.

Note: The NameValueLength and NameValueData fields are optional but, if present, they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required. For example:

```
length1 data1 length2 data2 length3 data3
```

As these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQRFH_STRUC_ID
Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined. This has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG)
The length in bytes of the MQRFH2 structure, including the NameValueLength and NameValueData fields at the end of the structure. It is valid for there to be multiple pairs of NameValueLength and NameValueData fields at the end of the structure, in the sequence:

```
length1, data1, length2, data2, ...
```

StrucLength does not include any user data that may follow the last NameValueData field at the end of the structure.

To avoid problems with converting the user data in some environments, StrucLength must be a multiple of four.

The constant shown here gives the length of the fixed part of the structure; that is, the length excluding the NameValueLength and NameValueData fields:

MQRFH_STRUC_LENGTH_FIXED_2
Length of fixed part of MQRFH2 structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED_2.

Version (MQLONG)
The structure version number. The value must be:

MQRFH_VERSION_2
Version-2 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_2.

Initial values and language declarations

Here are the initial values and language declarations for MQRFH2.

Table 64. Initial values of fields in MQRFH2

Field name	Name of constant	Value of constant
StrucId	MQRFH2_STRUC_ID	'RFH'
Version	MQRFH2_VERSION_1	2
StrucLength	MQRFH_STRUC_LENGTH_FIXED_2	36

Table 64. Initial values of fields in MQRFH2 (continued)

Field name	Name of constant	Value of constant
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_INHERIT	-2
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0
NameValueCCSID	None	1208

Note: In the C programming language, the macro variable MQRFH2_DEFAULT contains the values listed above. Issue this statement to provide initial values for the fields in the structure:

```
MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};
```

C declaration

```
struct tagMQRFH2 {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH2 including all
                               NameValueLength and NameValueData
                               fields */
  MQLONG   Encoding;        /* Numeric encoding of data that follows
                               last NameValueData field */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows last NameValueData field */
  MQCHAR8  Format;          /* Format name of data that follows last
                               NameValueData field */
  MQLONG   Flags;           /* Flags */
  MQLONG   NameValueCCSID;  /* Character set identifier of
                               NameValueData */
};
```

COBOL declaration

```
** MQRFH2 structure
 10  MQRFH.
** Structure identifier
 15  MQRFH-STRUCID                    PIC X(4).
** Structure version number
 15  MQRFH-VERSION                    PIC S9(9) COMP.
** Total length of MQRFH2 including all NameValueLength and
** NameValueData fields
 15  MQRFH-STRUCLENGTH                PIC S9(9) COMP.
** Numeric encoding of data that follows last NameValueData field
 15  MQRFH-ENCODING                   PIC S9(9) COMP.
** Character set identifier of data that follows last
** NameValueData field
 15  MQRFH-CODEDCHARSETID             PIC S9(9) COMP.
** Format name of data that follows last NameValueData field
 15  MQRFH-FORMAT                     PIC X(8).
** Flags
 15  MQRFH-FLAGS                      PIC S9(9) COMP.
** Character set identifier of NameValueData
 15  MQRFH-NAMEVALUECCSID             PIC S9(9) COMP.
```

PL/I declaration

```
dcl
 1 MQRFH2 based,
  3 StrucId           char(4)
    init(MQRFH_STRUC_ID),            /* Structure identifier */
  3 Version           fixed bin(31)
    init(MQRFH_VERSION_2),           /* Structure version number */
  3 StrucLength       fixed bin(31)
    init(MQRFH_STRUC_LENGTH_FIXED_2), /* Total length of MQRFH2 */
                                     /* including all */
                                     /* NameValueLength and */
                                     /* NameValueData fields */
  3 Encoding          fixed bin(31)
    init(MQENC_NATIVE),              /* Numeric encoding of data */
                                     /* that follows last */
                                     /* NameValueData field */
  3 CodedCharSetId    fixed bin(31)
    init(MQCCSI_INHERIT),            /* Character set identifier of */
                                     /* data that follows last */
                                     /* NameValueData field */
  3 Format            char(8)
    init(MQFMT_NONE),                /* Format name of data that */
                                     /* follows last NameValueData */
                                     /* field */
  3 Flags             fixed bin(31)
    init(MQRFH_NONE),                /* Flags */
  3 NameValueCCSID    fixed bin(31)
    init(1208);                      /* Character set identifier of */
                                     /* NameValueData */
```

MQRR – Response record

The MQRR structure is used to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list. MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, it is possible to determine the completion codes and reason codes for all of the queues in a distribution list when the outcome of the call is mixed. That is, when the call succeeds for some queues in the list, but fails for others. Reason code MQRC_MULTIPLE_REASONS from the call indicates that the response records, if provided by the application, have been set by the queue manager.

Fields

Here is a summary of the fields.

Table 65. Fields in MQRR

Field	Description
CompCode	Completion code
Reason	Reason code

Here is a description of the fields.

CompCode (MQLONG)
Completion code of queue.

This is the completion code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQRC_NONE.

Reason (MQLONG)
Reason code for queue.

This is the reason code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQRC_NONE.

C declaration

```
typedef struct tagMQRR MQRR;
struct tagMQRR
{
  MQLONG    CompCode;    /* Completion code for queue */
  MQLONG    Reason;      /* Reason code for queue */
} MQRR;
typedef MQRR MQPOINTER PMQRR;
```

COBOL declaration

```
  ** MQRR structure
   10 MQRR.
  ** Completionm code for queue
    15 MQRR-COMPCODE PIC S9(9) BINARY VALUE 0.
  ** Reason code for queue
    15 MQRR-REASON PIC S9(9) BINARY VALUE 0.
```

PL/I declaration

```
  dcl
   1 MQRR based,
    3 CompCode fixed bin(31)
        init(MQCC_OK),               /* Completion code for queue */
    3 Reason   fixed bin(31)
        init(MQRC_NONE);             /* Reason code for queue */
```

MQSD – Subscription descriptor

The MQSD structure is used to specify details about the subscription being made. The structure is an input/output parameter on the MQSUB call.

Fields

Here is a summary of the fields.

Table 66. Fields in MQSD

Field	Description
StrucId	Structure identifier
VersionStructure	Version number
Options	Options
ObjectName	Object name
AlternateUserId	Alternate User Id
AlternateSecurityId	Alternate Security ID
SubExpiry	Subscription Expiry
ObjectString	Object String

Table 66. Fields in MQSD (continued)

Field	Description
SubName	Subscription Name
SubUserData	Subscription user data
SubCorrelId	Subscription Correlation ID
PubPriority	Publication priority
PubAccountingToken	Publication Accounting Token
PubAppIdentityData	Publication application identity data
SelectionString	String providing selection criteria
SubLevel	Subscription Level
ResObjectString	Long object name

Here is a description of the fields.

StrucId (MQCHAR4)
This is the structure identifier; the value must be: MQSD_STRUC_ID Identifier for Subscription Descriptor structure.

For the C programming language, the constant MQSD_STRUC_ID_ARRAY is also defined; this has the same value as MQSD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSD_STRUC_ID.

Version (MQLONG)
This is the structure version number; the value must be:

MQSD_VERSION_1
Version-1 Subscription Descriptor structure.

Options (MQLONG)
This provides options to control the action of the MQSUB call. You must specify at least one of the following options:

- MQSO_RESUME
- MQSO_CREATE

MQSO_CREATE
Create a new subscription for the topic specified. If a subscription using the same SubName exists, the call fails with MQRC_SUB_ALREADY_EXISTS. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME. The SubName is not always necessary. For more details, see the description of that field.

Combining MQSO_CREATE with MQSO_RESUME returns a handle to a pre-existing subscription for the specified SubName if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

MQSO_RESUME
Return a handle to a pre-existing subscription which matches that specified by SubName. No changes are made to the matching subscriptions attributes and they are returned on output in the MQSD structure. Only the following MQSD fields are used: StrucId, Version, Options, AlternateUserId and AlternateSecurityId, and SubName.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME.

The user ID of the subscription is the user ID that created the subscription, or if it has been later altered by a different user ID, it is the user ID of the most recent successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the MQSO_ANY_USERID option, and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use, the call fails with MQRC_SUBSCRIPTION_IN_USE.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with MQRC_INVALID_SUBSCRIPTION. MQSO_RESUME is implied by MQSO_ALTER so you do not need to combine it with that option. However, combining the two options does not cause an error.

Durability options

The following options control how durable the subscription is. You can specify only one of these options. On return from an MQSUB call using MQSO_RESUME, the appropriate durability option is set.

MQSO_DURABLE

Request that the subscription to this topic remains until it is explicitly removed using MQCLOSE with the MQCO_REMOVE_SUB option. If this subscription is not explicitly removed it will remain even after this applications connection to the queue manager is closed. If a durable subscription is requested to a topic that is defined as not allowing durable subscriptions, the call fails with MQRC_DURABILITY_NOT_ALLOWED.

MQSO_NON_DURABLE

Request that the subscription to this topic is removed when the applications connection to the queue manager is closed, if it is not already explicitly removed. MQSO_NON_DURABLE is the opposite of the MQSO_DURABLE option, and is defined to aid program documentation. It is the default if neither is specified.

Destination options

The following option controls the destination that publications for a topic that has been subscribed to are sent to. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

MQSO_MANAGED

Request that the destination that the publications are sent to is managed by the queue manager. The object handle

returned in Hobj represents a queue manager managed queue and is for use with subsequent MQGET, MQINQ, or MQCLOSE calls. An object handle returned from a previous MQSUB call cannot be provided in the Hobj parameter when MQSO_MANAGED is not specified.

Scope option

The following option controls the scope of the subscription being made. On returning from an MQSUB call using MQSO-RESUME, the appropriate scope option is set.

MQSO_SCOPE_QMGR

This subscription is made only on the local queue manager. Note this is only value supported in z/VSE. Only publications that are published at this queue manager are sent to this subscriber.

Registration options

The following options control the details of the registration that is made to the queue manager for this subscription. On return from an MQSUB call using MQSO.

MQSO_ANY_USERID

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 causes the call to fail with MQRC_SUBSCRIPTION_IN_USE.

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. On successful completion, future publications to this subscriber are put to the subscribers queue with the new user ID set in the publication message.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can be altered or resumed by only the last user ID to alter the subscription.

If a user ID other than the one recorded as owning a subscription tries to resume the call fails with MQRC_IDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

Publication options

The following options control the way publications are sent to this subscriber.

MQSO_NEW_PUBLICATIONS_ONLY
No currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when MQSO_CREATE is specified. Any subsequent changes to a subscription do not alter the flow of publications and so any publications retained on a topic, will have already been sent to the subscriber as new publications.

If this option is specified without MQSO_CREATE the call fails with MQRC_OPTIONS_ERROR. On return from an MQSUB call using MQSO_RESUME, this option is not set even if the subscription was created using this option.

If this option is not used, previously retained messages are sent to the destination queue provided. If this action fails due to an error, either MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED, the creation of the subscription fails.

MQSO_PUBLICATIONS_ON_REQUEST
Setting this option indicates that the subscriber will request information specifically when required. The queue manager does not send unsolicited messages to the subscriber. The retained publication (or possibly multiple publications if a wildcard is specified in the topic) is sent to the subscriber each time an MQSUBRQ call is made using the Hsub handle from a previous MQSUB call. No publications are sent as a result of the MQSUB call using this option. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

ObjectName (MQCHAR48)
This is the name of the topic object as defined on the local queue manager. The name can contain the following characters:
- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks.

The ObjectName is used to form the full topic name.

The full topic name can be built from two different fields: ObjectName and ObjectString. For details of how these two fields are used, see “Using topic strings” on page 1077.

If the object identified by the ObjectName field cannot be found, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME even if there is a string specified in ObjectString.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged.

The length of this field is given by MQ_TOPIC_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

AlternateUserId (MQCHAR12)
Ignored in z/VSE

AlternateSecurityId (MQBYTE40)
Ignored in z/VSE

SubExpiry (MQLONG)
This is the time expressed in tenths of a second after which the subscription expires. No more publications will match this subscription after this interval has passed. As soon as a subscription expires, publications are no longer sent to the queue. However, the publications that are already there are not affected in any way. SubExpiry has no effect on publication expiry.

The following special value is recognized:

MQEI_UNLIMITED
The subscription has an unlimited expiration time.

On return from an MQSUB call using the MQSO_RESUME option this field is set to the original expiry of the subscription and not the remaining expiry time.

ObjectString (MQCHARV)
This is the long object name to be used.

The ObjectString is used to form the Full topic name.

The full topic name can be built from two different fields: ObjectName and ObjectString. For details of how these two fields are used, see "Using topic strings" on page 1077.

The maximum length of ObjectString is 256 in z/VSE.

If ObjectString is not specified correctly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR. This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged. The full topic name used is returned in the ResObjectString field if a buffer is provided.

SubName (MQCHARV)
This specifies the subscription name. This field is only required if Options specifies the option MQSO_DURABLE, but if provided will be used by the queue manager for MQSO_NON_DURABLE as well.

If specified, SubName must be unique within the queue manager, because it is the method used to identify the subscription.

The maximum length of SubName is 48 bytes in z/VSE.

This field serves two purposes. For an MQSO_DURABLE subscription, you use this field to identify a subscription so you can resume it after it has been created if you have either closed the handle to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. This is done using the MQSUB call with the MQSO_RESUME option. It is also displayed in the administrative view of subscriptions in the SUBNAME field in DISPLAY SBSTATUS.

If SubName is specified incorrectly, according to the description of how to use the MQCHARV structure, is left out when it is required (that is

SubName.VSLength is zero), or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_NAME_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

SubUserData (MQCHARV)
Ignored in z/VSE

SubCorrelId (MQBYTE24)
This field contains a correlation identifier common to all publications matching this subscription.

All publications sent to match this subscription contain this correlation identifier in the message descriptor. If multiple subscriptions get their publications from the same queue, using MQGET by correlation identifier allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

If the option MQSO_SET_CORREL_ID is not specified, the correlation identifier is generated by the queue manager and this field is an output field containing the correlation identifier that will be set in each message published for this subscription.

If the option MQSO_SET_CORREL_ID is specified, the correlation identifier is generated by the user and this field is an input field containing the correlation identifier to be set in each publication for this subscription. In this case, if the field contains MQCI_NONE, the correlation identifier that is set in each message published for this subscription is the correlation identifier created by the original put of the message.

The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current correlation identifier for the subscription.

PubPriority (MQLONG)
Ignored in z/VSE

PubAccountingToken (MQBYTE32)
Ignored in z/VSE

PubApplIdentityData (MQCHAR32)
Ignored in z/VSE

SelectionString (MQCHARV)
This is not supported in z/VSE. Value ignored.

SubLevel (MQLONG)
Ignored in z/VSE

ResObjectString (MQCHARV)
This is the long object name after the queue manager resolves the name provided in ObjectName.

If the long object name is provided in ObjectString and nothing is provided in ObjectName, then the value returned in this field is the same as provided in ObjectString.

If this field is omitted (that is ResObjectString.VSBufSize is zero) then the ResObjectString is not returned, but the length is returned in ResObjectString.VSLength. If the length is shorter than the full

ResObjectString then it is truncated and returns as many of the rightmost characters as can fit in the provided length.

If ResObjectString is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

MQSMPO – Set message property options

The MQSMPO structure allows applications to specify options that control how properties of messages are set. The structure is an input parameter on the MQSETMP call.

Data in MQSMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields

Here is a summary of the fields.

Table 67. Fields in MQSMPO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options
ValueEncoding	Property value encoding
ValueCCSID	Property value character set

Here is a description of the fields.

Options (MQLONG)

Location options: The following options relate to the relative location of the property compared to the property cursor.

MQSMPO_SET_FIRST

Sets the value of the first property that matches the specified name, or if it does not exist, adds a new property after all other properties with a matching hierarchy.

MQSMPO_SET_PROP_UNDER_CURSOR

Sets the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the MsgHandle field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_BEFORE_CURSOR
Sets a new property before the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the MsgHandle field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_AFTER_CURSOR
Sets a new property after the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the MsgHandle field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

If you need none of the options described, use this option:

MQSMPO_NONE
No options specified. This is always an input field. The initial value of this field is MQSMPO_SET_FIRST.

StrucId (MQCHAR4)
This is the structure identifier; the value must be:

MQSMPO_STRUC_ID
Identifier for set message property options structure.

For the C programming language, the constant MQSMPO_STRUC_ID_ARRAY is also defined. This has the same value as MQSMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSMPO_STRUC_ID.

ValueCCSID (MQLONG)
The character set of the property value to be set if the value is a character string.

This is always an input field. The initial value of this field is MQCCSI_APPL.

ValueEncoding (MQLONG)
The encoding of the property value to be set if the value is numeric.

This is always an input field. The initial value of this field is MQENC_NATIVE.

Version (MQLONG)
This is the structure version number; the value must be:

MQSMPO_VERSION_1
Version-1 set message property options structure.

This constant specifies the version number of the current version:

MQSMPO_CURRENT_VERSION
Current version of set message property options structure.

This is always an input field. The initial value of this field is MQSMPO_VERSION_1.

Initial values and language declarations

Here are the initial values and language declarations for MQSMPO.

Table 68. Initial values of fields in MQSMPO

Field name	Name of constant	Value of constant
StrucId	MQSMPO_STRUC_ID	'SMPO'
Version	MQSMPO_VERSION_1	1
Options	MQSMPO_NONE	0
ValueEncoding	MQENC_NATIVE	Depends on environment
ValueCCSID	MQCCSI_APPL	-3

Note:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQSMPO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSMPO MySMPO = {MQSMPO_DEFAULT};
```

C declaration

```
struct tagMQSMPO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQSETMP */
  MQLONG   ValueEncoding;  /* Encoding of Value */
  MQLONG   ValueCCSID;     /* Character set identifier of Value */
};
```

COBOL declaration

```
** MQSMPO structure
 10  MQSMPO.
** Structure identifier
 15  MQSMPO-STRUCID                    PIC X(4).
** Structure version number
 15  MQSMPO-VERSION                    PIC S9(9) BINARY.
** Options that control the action of MQSETMP
 15  MQSMPO-OPTIONS                    PIC S9(9) BINARY.
** Encoding of Value
 15  MQSMPO-VALUEENCODING              PIC S9(9) BINARY.
** Character set identifier of Value
 15  MQSMPO-VALUECCSID                 PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQSMPO based,
  3 StrucId             char(4)
    init(MQSMPO_STRUC_ID),       /* Structure identifier */
  3 Version             fixed bin(31)
    init(MQSMPO_VERSION_1),      /* Structure version number */
  3 Options             fixed bin(31)
    init(MQSMPO_SET_FIRST),      /* Options that control the action */
                                 /* of MQSETMP */
  3 ValueEncoding       fixed bin(31)
    init(MQENC_NATIVE),          /* Encoding of Value */
  3 ValueCCSID          fixed bin(31)
    init(MQCCSI_APPL);           /* Character set identifier of */
                                 /* Value */
```

MQSRO - Subscription request options

The MQSRO structure allows the application to specify options that control how a subscription request is made. The structure is an input/output parameter on the MQSUBRQ call.

Fields

Here is a summary of the fields.

Table 69. Fields in MQSRO

Field	Description
StrucId	Structure identifier
Version	Structure version number
Options	Options
NumPubs	Number of publications

Here is a description of the fields.

StrucId (MQCHAR4)
This is the structure identifier; the value must be: MQSRO_STRUC_ID Identifier for Subscription Request Options structure.

For the C programming language, the constant MQSRO_STRUC_ID_ARRAY is also defined; this has the same value as MQSRO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSRO_STRUC_ID.

Version (MQLONG)
This is the structure version number; the value must be: MQSRO_VERSION_1 Version-1 Subscription Request Options structure.

The following constant specifies the version number of the current version: MQSRO_CURRENT_VERSION Current version of Subscription Request Options structure.

This is always an input field. The initial value of this field is MQSRO_VERSION_1.

Options (MQLONG)
Default option: the following option must be used:

MQSRO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQSRO_NONE helps program documentation.

Although it is not intended that this option be used with any other, because its value is zero, this use cannot be detected.

NumPubs (MQLONG)
This is an output field, returned to the application to indicate the number of publications sent to the subscription queue as a result of this call. Although this number of publications have been sent as a result of this call, there is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There might be more than one publication if the topic subscribed to contained a wildcard. If no wildcards were present in the topic string when the subscription represented by Hsub was created, then at most one publication is sent as a result of this call.

MQTM – Trigger message

On the WebSphere MQ for z/VSE platform, triggers are invoked by the queue manager using either the transaction ID code, or program ID code, in the queue definition.

The transaction ID, or program ID, determines if the trigger is invoked using an EXEC CICS START, or EXEC CICS LINK, program respectively.

Trigger programs using the START mechanism can use the EXEC CICS RETRIEVE program to retrieve the trigger data structure. Programs invoked by the LINK mechanism can retrieve the MQTM structure in the DFHCOMMAREA.

Fields

Here is a summary of the fields.

Table 70. Fields in MQTM

Field	Description
StrucId	Structure identifier
Version	Structure version number
QName	Name of triggered queue
ProcessName	Name of process object
TriggerData	Trigger data
ApplType	Application type
ApplId	Application identifier

Table 70. Fields in MQTM (continued)

Field	Description
EnvData	Environment data
UserData	User data

Here is a description of the fields.

ApplId (MQCHAR256)
Application identifier.

On WebSphere MQ for z/VSE ApplId is:
- A CICS transaction ID (for MQAT_CICS_VSE applications).

ApplType (MQLONG)
Application type.

On WebSphere MQ for z/VSE ApplType has the following standard value:
MQAT_CICS_VSE
WebSphere MQ for z/VSE application.

The initial value of this field is `0`.

EnvData (MQCHAR128)
Environment data.

This is a reserved field.

ProcessName (MQCHAR48)
Name of process object.

This is a reserved field.

QName (MQCHAR48)
This is the name of the queue for which a trigger event occurred, and is used by the application started by the trigger-monitor application. The queue manager initializes this field with the value of the QName attribute of the triggered queue.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQTM_STRUC_ID
Identifier for trigger message structure

For the C programming language, the constant MQTM_STRUC_ID_ARRAY is also defined; this has the same value as MQTM_STRUC_ID, but is an array of characters instead of a string.The initial value of this field is MQTM_STRUC_ID.

TriggerData (MQCHAR64)
Trigger data.

On WebSphere MQ for z/VSE this is a 13-byte field, consisting of:
- 4-byte transaction ID code.
- 8-byte program ID code.

- 1-byte trigger-event flag.

The trigger event flag indicates whether the trigger was started from a (F)irst message or (E)very message event. When a trigger instance is a program (rather than a transaction), the MQTM data structure is passed to the trigger program as a COMMAREA.

In the case of (E)very event triggers, the queue manager will start another trigger instance when a trigger instance ends, if it detects that there are still messages on the queue.

If an error has occurred such that the trigger program cannot process messages from the object queue, a loop condition may arise (the queue manager will continue to start trigger instances which themselves continue to fail).

This can be avoided by setting the trigger event flag in the TriggerData field of the MQTM data structure to "S" (stop). Since the trigger event flag is in a COMMAREA, the queue manager will detect that the event flag has been set to (S)top, and will cease starting new trigger instances.

UserData (MQCHAR128)
User data.

Contains user-defined data configured in the queue definition of the queue that initiates the trigger instance.

Version (MQLONG)
Structure version number.

The value must be:

MQTM_VERSION_1
Version number for trigger message structure.

C declaration

```
typedef struct tagMQTM MQTM;
struct tagMQTM
{
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQCHAR48  QName;           /* Name of triggered queue */
  MQCHAR48  ProcessName;     /* Name of process object */
  MQCHAR64  TriggerData;     /* Trigger data */
  MQLONG    ApplType;        /* Application type */
  MQCHAR256   ApplId;        /* Application identifier */
  MQCHAR128   EnvData;       /* Environment data */
  MQCHAR128   UserData;      /* User data */
};
```

COBOL declaration

```
** MQTM structure
 10 MQTM.
** Structure identifier
   15 MQTM-STRUCID PIC X(4).
** Structure version number
   15 MQTM-VERSION PIC S9(9) BINARY.
** Name of triggered queue
   15 MQTM-QNAME PIC X(48).
** Name of process object
   15 MQTM-PROCESSNAME PIC X(48).
** Trigger data
   15 MQTM-TRIGGERDATA PIC X(64).
** Application type
   15 MQTM-APPLTYPE PIC S9(9) BINARY.
** Application identifier
   15 MQTM-APPLID PIC X(256).
** Environment data
   15 MQTM-ENVDATA PIC X(128).
** User data
   15 MQTM-USERDATA PIC X(128).
```

PL/I declaration

```
dcl
 1 MQTM based,
   3 StrucId char(4),               /* Structure identifier */
   3 Version fixed bin(31),         /* Structure version number */
   3 QName char(48),                /* Name of triggered queue */
   3 ProcessName char(48),          /* Name of process object */
   3 TriggerData char(64),          /* Trigger data */
   3 ApplType fixed bin(31),        /* Application type */
   3 ApplId char(256),              /* Application identifier */
   3 EnvData char(128),             /* Environment data */
   3 UserData char(128);            /* User data */
```

MQXQH – Transmission-queue header

The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues. A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the Usage queue attribute having the value MQUS_TRANSMISSION.

A message that is on a transmission queue has two message descriptors:

- One message descriptor is stored separately from the message data; this is called the separate message descriptor, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call (see below for details). The separate message descriptor is the one that is returned to the application in the MsgDesc parameter of the MQGET call when the message is removed from the transmission queue.
- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the embedded message descriptor, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations; see below for details).

  The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

  – Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or

– Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

The embedded message descriptor is the one that is returned to the application in the MsgDesc parameter of the MQGET call when the message is removed from the final destination queue.

Putting messages on remote queues

When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:

- Creates an MQXQH structure containing the embedded message descriptor.
- Appends an MQMDE if one is needed and is not already present.
- Appends the application message data.
- Places the message on an appropriate transmission queue.

Putting messages directly on transmission queues

An application can also put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure, and initialize the fields with appropriate values. In addition, the Format field in the MsgDesc parameter of the MQPUT or MQPUT1 call must have the value MQFMT_XMIT_Q_HEADER.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the CodedCharSetId queue-manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

On z/VSE, the queue manager does check that an MQXQH structure is present, but does not check that valid values have been specified for the fields.

Getting messages from transmission queues

Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value MQFMT_XMIT_Q_HEADER being returned in the Format field in the MsgDesc parameter of the MQGET call. The values returned in the CodedCharSetId and Encoding fields in the MsgDesc parameter indicate the character set and encoding of the character and integer data in the MQXQH structure, respectively. The character set and encoding of the application message data are defined by the CodedCharSetId and Encoding fields in the embedded message descriptor.

Fields

Here is a summary of the fields.

Table 71. Fields in MQXQH

Field	Description
StrucId	Structure identifier

Table 71. Fields in MQXQH (continued)

Field	Description
Version	Structure version number
RemoteQName	Name of destination queue
RemoteQMgrName	Name of destination queue manager
MsgDesc	Original message descriptor

Here is a description of the fields.

MsgDesc (MQMD1)
This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the MsgDesc parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD. The initial values of the fields in this structure are the same as those in the MQMD structure.

RemoteQMgrName (MQCHAR48)
This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

RemoteQName (MQCHAR48)
This is the name of the message queue that is the apparent eventual destination for the message (this might prove not to be the eventual destination if, for example, this queue is defined at RemoteQMgrName to be a local definition of another remote queue).

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)
This is the structure identifier. The value must be:

MQXQH_STRUC_ID
Identifier for transmission-queue header structure

For the C programming language, the constant MQXQH_STRUC_ID_ARRAY is also defined; this has the same value as MQXQH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXQH_STRUC_ID.

Version (MQLONG)
This is the structure version number. The value must be:

MQXQH_VERSION_1
Version number for transmission-queue header structure.

The following constant specifies the version number of the current version:

MQXQH_CURRENT_VERSION
Current version of transmission-queue header structure.

The initial value of this field is MQXQH_VERSION_1.

C declaration

```
typedef struct tagMQXQH MQXQH;
struct tagMQXQH
{
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQCHAR48  RemoteQName;     /* Name of destination queue */
  MQCHAR48  RemoteQMgrName;  /* Name of destination queue manager */
  MQMD1     MsgDesc;         /* Original message descriptor */
};
```

COBOL declaration

```
  ** MQXQH structure
   10 MQXQH.
  ** Structure identifier
    15 MQXQH-STRUCID PIC X(4).
  ** Structure version number
    15 MQXQH-VERSION PIC S9(9) BINARY.
  ** Name of destination queue
    15 MQXQH-REMOTEQNAME PIC X(48).
  ** Name of destination queue manager
    15 MQXQH-REMOTEQMGRNAME PIC X(48).
  ** Original message descriptor
    15 MQXQH-MSGDESC.
  ** Structure identifier
       20 MQXQH-MSGDESC-STRUCID PIC X(4).
  ** Structure version number
     20 MQXQH-MSGDESC-VERSION PIC S9(9) BINARY.
  ** Report options
     20 MQXQH-MSGDESC-REPORT PIC S9(9) BINARY.
  ** Message type
     20 MQXQH-MSGDESC-MSGTYPE PIC S9(9) BINARY.
  ** Expiry time
     20 MQXQH-MSGDESC-EXPIRY PIC S9(9) BINARY.
  ** Feedback or reason code
     20 MQXQH-MSGDESC-FEEDBACK PIC S9(9) BINARY.
  ** Numeric encoding of message data
     20 MQXQH-MSGDESC-ENCODING PIC S9(9) BINARY.
  ** Character set identifier of message data
     20 MQXQH-MSGDESC-CODEDCHARSETID PIC S9(9) BINARY.
  ** Format name of message data
     20 MQXQH-MSGDESC-FORMAT PIC X(8).
  ** Message priority
     20 MQXQH-MSGDESC-PRIORITY PIC S9(9) BINARY.
  ** Message persistence
     20 MQXQH-MSGDESC-PERSISTENCE PIC S9(9) BINARY.
  ** Message identifier
     20 MQXQH-MSGDESC-MSGID PIC X(24).
  ** Correlation identifier
     20 MQXQH-MSGDESC-CORRELID PIC X(24).
  ** Backout counter
     20 MQXQH-MSGDESC-BACKOUTCOUNT PIC S9(9) BINARY.
  ** Name of reply-to queue
     20 MQXQH-MSGDESC-REPLYTOQ PIC X(48).
  ** Name of reply queue manager
     20 MQXQH-MSGDESC-REPLYTOQMGR PIC X(48).
  ** User identifier
     20 MQXQH-MSGDESC-USERIDENTIFIER PIC X(12).
  ** Accounting token
     20 MQXQH-MSGDESC-ACCOUNTINGTOKEN PIC X(32).
  ** Application data relating to identity
     20 MQXQH-MSGDESC-APPLIDENTITYDATA PIC X(32).
  ** Type of application that put the message
     20 MQXQH-MSGDESC-PUTAPPLTYPE PIC S9(9) BINARY.
  ** Name of application that put the message
     20 MQXQH-MSGDESC-PUTAPPLNAME PIC X(28).
  ** Date when message was put
     20 MQXQH-MSGDESC-PUTDATE PIC X(8).
  ** Time when message was put
     20 MQXQH-MSGDESC-PUTTIME PIC X(8).
  ** Application data relating to origin
     20 MQXQH-MSGDESC-APPLORIGINDATA PIC X(4).
```

PL/I declaration

```
dcl
 1 MQXQH based,
  3 StrucId char(4),                /* Structure identifier */
  3 Version fixed bin(31),          /* Structure version number */
  3 RemoteQName char(48),           /* Name of destination queue */
  3 RemoteQMgrName char(48),        /* Name of destination queue manager */
  3 MsgDesc,                        /* Original message descriptor */
   5 StrucId char(4),               /* Structure identifier */
   5 Version fixed bin(31),         /* Structure version number */
   5 Report fixed bin(31),          /* Report options */
   5 MsgType fixed bin(31),         /* Message type */
   5 Expiry fixed bin(31),          /* Expiry time */
   5 Feedback fixed bin(31),        /* Feedback or reason code */
   5 Encoding fixed bin(31),        /* Numeric encoding of message data */
   5 CodedCharSetId fixed bin(31),  /* Character set identifier of message data */
   5 Format char(8),                /* Format name of message data */
   5 Priority fixed bin(31)         /* Message priority */
   5 Persistence fixed bin(31),     /* Message persistence */
   5 MsgId char(24),                /* Message identifier */
   5 CorrelId char(24),             /* Correlation identifier */
   5 BackoutCount fixed bin(31),    /* Backout counter */
   5 ReplyToQ char(48),             /* Name of reply-to queue */
   5 ReplyToQMgr char(48),          /* Name of reply queue manager */
   5 UserIdentifier char(12),       /* User identifier */
   5 AccountingToken char(32),      /* Accounting token */
   5 ApplIdentityData char(32),     /* Application data relating to identity */
   5 PutApplType fixed bin(31),     /* Type of application that put the message */
   5 PutApplName char(28),          /* Name of application that put the message */
   5 PutDate char(8),               /* Date when message was put */
   5 PutTime char(8),               /* Time when message was put */
   5 ApplOriginData char(4);        /* Application data relating to origin */
```

MQI calls

This section identifies the MQI calls supported by WebSphere MQ for z/VSE, which are:

MQBACK
: Back out changes

MQBUFMH
: Convert buffer into message handle

MQCLOSE
: Close object

MQCMIT
: Commit changes

MQCONN
: Connect queue manager

MQCRTMH
: Create message handle

MQDISC
: Disconnect queue manager

MQDLTMH
: Delete message handle

MQDLTMP
: Delete message properties

MQGET
: Get message

MQINQ
: Inquire about object attributes

MQINQMP
: Inquire message property

MQMHBUF
: Comvert message handle into buffer

MQOPEN
Open object
MQPUT
Put message
MQPUT1
Put one message
MQSET
Set object attributes
MQSETMP
Set message property
MQSUB
Register subscription
MQSUBRQ
Request retained subscription

The MQCONNX MQI call is not supported by WebSphere MQ for z/VSE, however the WebSphere MQ for z/VSE server will accept connection requests by MQ client programs issuing the MQCONNX call. In addition, the WebSphere MQ client for z/VSE supports the MQCONNX call. The client is described in Chapter 10, "WebSphere MQ clients," on page 623.

MQBACK - Back out changes

The MQBACK call indicates to the queue manager that all of the message gets and puts that have occurred since the last syncpoint are to be backed out (for client and batch programs) or have been backed out (for online programs). Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue (for client and batch programs).

Note: For online applications, it is required that the application itself performs a CICS SYNCPOINT ROLLBACK before issuing the MQBACK call. The MQBACK call is used to re-enter the queue manager to update internal WebSphere MQ for z/VSE tables.

For client and batch programs, the CICS SYNCPOINT ROLLBACK is performed for the caller.

Syntax

```
MQBACK (Hconn,CompCode,Reason)
```

Parameters

The MQBACK call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQBACK (Hconn,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBACK' USING HCONN,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQBACK (HCONN,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31); /*Connection handle */
DCL COMPCODE    FIXED BIN(31); /*Completion code */
DCL REASON      FIXED BIN(31); /*Reason code qualifying CompCode */
```

MQBUFMH - Convert buffer into message handle

The MQBUFMH function call converts a buffer into a message handle and is the inverse of the MQMHBUF call.

This call takes a message descriptor and MQRFH2 properties in the buffer and makes them available through a message handle. The MQRFH2 properties in the message data are, optionally, removed. The Encoding, CodedCharSetId, and Format fields of the message descriptor are updated, if necessary, to correctly describe the contents of the buffer after the properties have been removed.

Syntax

```
MQBUFMH (Hconn, Hmsg, BufMsgHOpts, MsgDesc, Buffer, BufferLength,
DataLength, CompCode, Reason)
```

Parameters

The MQBUFMH call has the following parameters:

Hconn (MQHCONN) - input
This handle represents the connection to the queue manager. The value of Hconn must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input
This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

BufMsgHOpts (MQBMHO) - input
The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. See "MQBMHO – Buffer to message handle options" on page 718 for details.

MsgDesc (MQMD) - input/output
The MsgDesc structure contains the message descriptor properties and describes the contents of the buffer area.

On output from the call, the properties are optionally removed from the buffer area and, in this case, the message descriptor is updated to correctly describe the buffer area.

Data in this structure must be in the character set and encoding of the application.

BufferLength (MQLONG) - input
BufferLength is the length of the Buffer area, in bytes.

A BufferLength of zero bytes is valid, and indicates that the buffer area contains no data.

Buffer (MQBYTExBufferLength) - input/output
Buffer defines the area containing the message buffer. For most data, you should align the buffer on a 4-byte boundary.

If Buffer contains character or numeric data, set the CodedCharSetId and Encoding fields in the MsgDesc parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the BufferLength parameter is zero, Buffer is not referred to; in this case, the parameter address passed by programs written in C or System/390® assembler can be null.

DataLength (MQLONG) - output
DataLength is the length, in bytes, of the buffer which might have the properties removed.

CompCode (MQLONG) - output
The completion code; it is one of the following:
MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE (0, X'000')
No reason to report.

If CompCode is MQCC_FAILED:

MQRC_BMHO_ERROR (2489, X'09B9')
Buffer to message handle options structure not valid.

MQRC_BUFFER_ERROR (2004, X'07D4')
Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR (2005, X'07D5')
Buffer length parameter not valid.

MQRC_CONNECTION_BROKEN (2009, X'07D9')
Connection to queue manager lost.

MQRC_HMSG_ERROR (2460, X'099C')
Message handle not valid.

MQRC_MD_ERROR (2026, X'07EA')
Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE (2499, X'09C3')
Message handle already in use.

MQRC_OPTIONS_ERROR (2046, X'07FE')
Options not valid or not consistent.

MQRC_RFH_ERROR (2334, X'091E')
MQRFH2 structure not valid.

MQRC_RFH_FORMAT_ERROR (2421, X'0975')
An MQRFH2 folder containing properties could not be parsed.

MQRC_UNEXPECTED_ERROR (2195, X'893')
Unexpected error occurred.

Usage notes

MQBUFMH calls cannot be intercepted by API exits. A buffer is converted into a message handle in the application space; the call does not reach the queue manager.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQBUFMH (  Hconn, Hmsg, &BufMsgHOpts, &MsgDesc, BufferLength,
Buffer,     &DataLength, &CompCode, &Reason);
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQBMHO BufMsgHOpts; /* Options that control the action of MQBUFMH */
MQMD MsgDesc; /* Message descriptor */
```

```
MQLONG BufferLength; /* Length in bytes of the Buffer area */
MQBYTE Buffer[n]; /* Area to contain the message buffer */
MQLONG DataLength; /* Length of the output buffer */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBUFMH' USING HCONN, HMSG, BUFMSGHOPTS, MSGDESC,
BUFFERLENGTH, BUFFER, DATALENGTH,COMPCODE, REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQBUFMH
01 BUFMSGHOPTS.
   COPY CMQBMHOV.
** Message descriptor
01 MSGDESC.
   COPY CMQMD.
** Length in bytes of the Buffer area
01 BUFFERLENGTH PIC S9(9) BINARY.
** Area to contain the message buffer
01 BUFFER PIC X(n).
** Length of the output buffer
01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBUFMH ( Hconn, Hmsg, BufMsgHOpts, MsgDesc, BufferLength,
Buffer,DataLength, CompCode, Reason);
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8) /* Message handle */
dcl BufMsgHOpts like MQBMHO;
                      /* Options that control the action of MQBUFMH */
dcl MsgDesc like MQMD; /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer char(n); /* Area to contain the message buffer */
dcl DataLength fixed bin(31); /* Length of the output buffer */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQCLOSE - Close object

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN call.

Syntax

```
MQCLOSE (Hconn,Hobj,Options,CompCode,Reason)
```

Parameters

The MQCLOSE call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

Hobj (MQHOBJ) - input/output
Object handle.

This handle represents the object that is being closed. The object can be of any type. The value of Hobj was returned by a previous MQOPEN call. On successful completion of the call, the queue manager sets this parameter to a value of binary zeros.

Options (MQLONG) - input
Options that control the action of MQCLOSE.

The following options are supported:

MQCO_NONE
No optional close processing required. This must be specified for:

- Objects other than queues.
- Predefined queues.
- Temporary dynamic queues (but only in those cases where Hobj is not the handle returned by the MQOPEN call that created the queue).

In all of the above cases, the object is retained and not deleted. If this option is specified for a temporary dynamic queue: the queue is deleted, if it was created by the MQOPEN call that returned Hobj; any messages that are on the queue are purged. In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

MQCO_DELETE
Delete the queue.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, and there are no messages on the queue.
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj. In this case, all the messages on the queue are purged.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

MQCO_DELETE_PURGE
Delete the queue, purging any messages on it.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue.
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

The following points apply if the object being closed is a dynamic queue (either permanent or temporary):

- For a dynamic queue, the options MQCO_DELETE or MQCO_DELETE_PURGE can be specified regardless of the options specified on the corresponding MQOPEN call.
- When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. After a dynamic queue has been deleted, any call (other than MQCLOSE) that attempts to reference the queue using a previously acquired Hobj handle fails with reason code MQRC_Q_DELETED.

  Be aware that although a deleted queue cannot be accessed by applications, the queue is not removed from the system, and associated resources are not freed, until such time as all handles that reference the queue have been closed.
- When a dynamic queue is deleted, if the Hobj handle specified on the MQCLOSE call is not the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier is authorized to delete the queue. This check is not performed if:
  - The WebSphere MQ for z/VSE security feature is not active.
  - The handle specified is the one returned by the MQOPEN call that created the queue.
  - The queue being deleted is a temporary dynamic queue.
- When a temporary dynamic queue is closed, if the Hobj handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated. If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but this does not cause the units of work to fail. However, as described above, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

CompCode (MQLONG) - output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.
MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') Option inconsistent with queue status.
MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQCLOSE (Hconn,&Hobj,Options,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQHOBJ  Hobj;           /*Object handle */
MQLONG  Options;        /*Options that control the action of MQCLOSE */
MQLONG  CompCode;       /*Completion code */
MQLONG  Reason;         /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCLOSE'USING HCONN,HOBJ,OPTIONS,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object handle
  01 HOBJ PICS9(9)BINARY.
**Options that control the action of MQCLOSE
  01 OPTIONS PICS9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQCLOSE (HCONN,HOBJ,OPTIONS,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31);  /*Connection handle */
DCL HOBJ        FIXED BIN(31);  /*Object handle */
DCL OPTIONS     FIXED BIN(31);  /*Options that control the action of
                                  MQCLOSE */
DCL COMPCODE    FIXED BIN(31);  /*Completion code */
DCL REASON      FIXED BIN(31);  /*Reason code qualifying CompCode */
```

MQCMIT - Commit changes

For client and batch programs, the MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and that all of the message gets and puts that have occurred since the last syncpoint are to be made permanent by the queue manager issuing a CICS SYNCPOINT for the application. Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

For online application programs, the MQCMIT call (issued after the online application issues a CICS SYNCPOINT) indicates to the queue manager it can verify that syncpoint has occurred, update internal tables and clear any delayed gets.

Syntax

```
MQCMIT (Hconn,CompCode,Reason)
```

Parameters

The MQCMIT call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

CompCode (MQLONG) - output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQCMIT (Hconn,&CompCode,&Reason)
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCMIT'USING HCONN,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQCMIT (HCONN,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31);  /*Connection handle */
DCL COMPCODE    FIXED BIN(31);  /*Completion code */
DCL REASON      FIXED BIN(31);  /*Reason code qualifying CompCode */
```

MQCONN - Connect queue manager

The MQCONN call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent message queuing calls.

Syntax

```
MQCONN (QMgrName,Hconn,CompCode,Reason)
```

Parameters

The MQCONN call has the following parameters:

QMgrName (MQCHAR48) - input
Name of queue manager.

The name specified must be the name of a local queue manager. In WebSphere MQ for z/VSE there is only one queue manager in a CICS partition.

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. It must be specified on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the CICS transaction terminates.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:
MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'7E9') Maximum number of connections reached.
MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQCONN (QMgrName,&Hconn,&CompCode,&Reason);
```

Declare the parameters:

```
MQCHAR48 QMgrName;      /*Name of queue manager */
MQHCONN Hconn;          /*Connection handle */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONN'USING QMGRNAME,HCONN,COMPCODE,REASON.
```

Declare the parameters:

```
**Name of queue manager
  01 QMGRNAME PICX(48).
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQCONN (QMGRNAME,HCONN,COMPCODE,REASON);
```

Declare the parameters:

```
DCL QMGRNAME    CHAR(48);       /*Name of queue manager */
DCL HCONN       FIXED BIN(31);  /*Connection handle */
DCL COMPCODE    FIXED BIN(31);  /*Completion code */
DCL REASON      FIXED BIN(31);  /*Reason code qualifying CompCode */
```

MQCRTMH - Create message handle

The MQCRTMH call returns a message handle. An application can use it on subsequent message queuing calls:

- Use the MQSETMP call to set a property of the message handle.
- Use the MQINQMP call to inquire on the value of a property of the message handle.
- Use the MQDLTMP call to delete a property of the message handle.

The message handle can be used on the MQPUT and MQPUT1 calls to associate the properties of the message handle with those of the message being put. Similarly, by specifying a message handle on the MQGET call, the properties of the message being retrieved can be accessed using the message handle when the MQGET call completes.

Use MQDLTMH to delete the message handle.

Syntax

```
MQCRTMH (Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason)
```

Parameters

The MQCRTMH call has the following parameters:

Hconn (MQHCONN) - input

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN or MQCONNX call. If the connection to the queue manager ceases to be valid and no WebSphere MQ call is operating on the message handle, MQDLTMH is implicitly called to delete the message.

Alternatively, you can specify this value:

CrtMsgHOpts (MQCMHO) - input
The options that control the action of MQCRTMH. See MQCMHO for details.

Hmsg (MQHMSG) - output

On output, a message handle is returned that can be used to set, inquire and delete properties of the message handle. Initially the message handle contains no properties.

A message handle also has an associated message descriptor. Initially this contains the default values. The values of the associated message descriptor fields can be set and inquired using the MQSETMP and MQINQMP calls. The MQDLTMP call resets a field of the message descriptor back to its default value.

The returned message handle can only be used on the specified connection.

The same Hconn parameter value must be used on the subsequent MQI calls where this message handle is used:

MQDLTMH MQSETMP MQINQMP MQDLTMP MQMHBUF MQBUFMH

The returned message handle ceases to be valid when the MQDLTMH call is issued for the message handle, or when the unit of processing that defines the scope of the handle terminates.

CompCode (MQLONG) - output

The completion code; it is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output

The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:

MQRC_CMHO_ERROR
(2461, X'099D') Create message handle options structure not valid.

MQRC_CONNECTION_BROKEN
(2273, X'7D9') Connection to queue manager lost.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'07E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQCRTMH (Hconn, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQCMHO CrtMsgHOpts; /* Options that control the action of MQCRTMH */
MQHMSG Hmsg; /* Message handle */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCRTMH' USING HCONN, CRTMSGOPTS, HMSG, COMPCODE, REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Options that control the action of MQCRTMH
01 CRTMSGHOPTS.
     COPY CMQCMHOV.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCRTMH (Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason);
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl CrtMsgHOpts like MQCMHO;
                        /* Options that control the action of MQCRTMH */
dcl Hmsg char(8); /* Message handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQDISC - Disconnect queue manager

The MQDISC call breaks the connection between the queue manager and the application program. It is the inverse of the MQCONN call.

Syntax

```
MQDISC (Hconn,CompCode,Reason)
```

Parameters

The MQDISC call has the following parameters:

Hconn (MQHCONN) - input/output
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

On successful completion of the call, the queue manager sets Hconn to binary zeros.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_FAILED:
MQRC_ADAPTER_DISC_LOAD_ERROR
(2138, X'85A') Unable to load adapter disconnection module.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQDISC (&Hconn,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDISC'USING HCONN,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN    PIC S9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON   PIC S9(9)BINARY.
```

PL/I invocation

```
CALL MQDISC (HCONN,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31);  /*Connection handle */
DCL COMPCODE    FIXED BIN(31);  /*Completion code */
DCL REASON      FIXED BIN(31);  /*Reason code qualifying CompCode *
/
```

MQDLTMH - Delete message handle

The MQDLTMH call deletes a message handle and is the inverse of the MQCRTMH call.

Syntax

```
MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason)
```

Parameters

The MQDLTMH call has the following parameters:

Hconn (MQHCONN) - input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input/output

This is the message handle to be deleted. The value was returned by a previous MQCRTMH call. On successful completion of the call, the handle is set to an invalid value for the environment. This value is:

MQHM_UNUSABLE_HMSG
Unusable message handle.

DltMsgHOpts (MQDMHO) - input
See MQDMHO for details.

CompCode (MQLONG) - output

The completion code. It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output

The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMHO_ERROR
(2462, X'099E') Delete message handle options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQDLTMH (Hconn, &Hmsg, &DltMsgHOpts, &CompCode, &Reason);
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQDMHO DltMsgHOpts; /* Options that control the action of MQDLTMH */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMH' USING HCONN, HMSG, DLTMSGOPTS, COMPCODE, REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQDLTMH
01 DLTMSGHOPTS.
COPY CMQDMHOV.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason);
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8); /* Message handle */
dcl DltMsgHOpts like MQDMHO;
                  /* Options that control the action of MQDLTMH */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQDLTMP - Delete message property

The MQDLTMP call deletes a property from a message handle and is the inverse of the MQSETMP call.

Syntax

```
MQDLTMP (Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason)
```

Parameters

The MQDLTMP call has the following parameters:

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input

This is the message handle containing the property to be deleted. The value was returned by a previous MQCRTMH call.

DltPropOps (MQDMPO) - Input

See the MQDMPO data type for details.

Name (MQCHARV) - input

The name of the property to delete.

Wildcards are not allowed in the property name.

CompCode (MQLONG) - output

The completion code. It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_WARNING:

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If CompCode is MQCC_FAILED:

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMPO_ERROR
(2481, X'09B1') Delete message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQDLTMP (Hconn, Hmsg, &DltPropOpts, &Name, &CompCode, &Reason)
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQDMPO DltPropOpts; /* Options that control the action of MQDLTMP */
MQCHARV Name; /* Property name */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMP' USING   HCONN, HMSG, DLTPROPOPTS, NAME, COMPCODE,
REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQDLTMP
01 DLTPROPOPTS.
COPY CMQDMPOV.
** Property name
01 NAME
COPY CMQCHRVV.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMP ( Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason );
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8); /* Message handle */
dcl DltPropOpts like MQDMPO;
                 /* Options that control the action of MQDLTMP */
dcl Name like MQCHARV; /* Property name */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQGET - Get message

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax

```
MQGET (Hconn,Hobj,MsgDesc,GetMsgOpts,BufferLength,
Buffer,DataLength,CompCode,Reason)
```

Parameters

The MQGET call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

Hobj (MQHOBJ) - input
Object handle.

This handle represents the queue from which a message is to be retrieved. The value of Hobj was returned by a previous MQOPEN call. The queue must have been opened with one or more of the following options (see "MQOPEN - Open object" on page 914 for details):
MQOO_INPUT_SHARED
MQOO_INPUT_EXCLUSIVE
MQOO_BROWSE

MsgDesc (MQMD) - input/output
Message descriptor.

This structure describes the attributes of the message required, and the attributes of the message retrieved. See "MQMD – Message descriptor" on page 774 for details.

If BufferLength is less than the message length, MsgDesc is still filled in by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the GetMsgOpts parameter (see the Options field described in "MQGMO – Get message options" on page 740.

GetMsgOpts (MQGMO) - input/output
Options that control the action of MQGET. See "MQGMO – Get message options" on page 740 for details.

BufferLength (MQLONG) - input
Length in bytes of the Buffer area.

Zero can be specified for messages that have no data, or if the message is to be removed from the queue and the data discarded (MQGMO_ACCEPT_TRUNCATED_MSG must be specified in this case).

Buffer (MQBYTE×BufferLength) - output
Area to contain the message data.

If BufferLength is less than the message length, as much of the message as possible is moved into Buffer ; this happens whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the GetMsgOpts parameter (see the Options field described in "MQGMO – Get message options" on page 740).

The character set and encoding of the data in Buffer are given (respectively) by the CodedCharSetId and Encoding fields returned in the MsgDesc parameter. If these are different from the values required by the

receiver, the receiver must convert the application message data to the character set and encoding required. The MQGMO_CONVERT option can be used with a user-written exit to perform the conversion of the message data (see "MQGMO – Get message options" on page 740 for details of this option).

Note: All of the other parameters on the MQGET call are in the character set of the local queue manager.

DataLength (MQLONG) - output
Length of the message.

This is the length in bytes of the application data in the message. If this is greater than BufferLength, only BufferLength bytes are returned in the Buffer parameter (that is, the message is truncated). If the value is zero, it means that the message contains no application data.

If BufferLength is less than the message length, DataLength is still filled in by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the GetMsgOpts parameter (see the Options field described in "MQGMO – Get message options" on page 740 for more information).

This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the MQGMO_CONVERT option is specified, and the converted message data is too long to fit in Buffer , the value returned for DataLength is:

- The length of the unconverted data, for queue-manager defined formats. In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer somewhat bigger than the value returned by the queue manager for DataLength.
- The value returned by the data-conversion exit, for application-defined formats.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

The reason codes listed below are the ones that the queue manager can return for the Reason parameter. If the application specifies the MQGMO_CONVERT option, and a user-written exit is invoked to convert some or all of the message data, it is the exit that decides what value is returned for the Reason parameter. As a result, values other than those documented below are possible.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_WARNING:

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).
MQRC_TRUNCATED_MSG_FAILED
(2080, X'820') Truncated message returned (processing not completed).

If CompCode is MQCC_FAILED:
MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too big for buffer.
MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.
MQRC_FILE_SYSTEM_ERROR
(2216, X'8A8') Queuer received file error.
MQRC_FORMAT_ERROR
(2110, X'83E') Format field not valid.
MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.
MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.
MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.
MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.
MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.
MQRC_NOT_CONVERTED
(2119, X'847') Application message data not converted.
MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.
MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.
MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.
MQRC_Q_DELETED
(2052, X'804') The queue has been deleted.
MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.
MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.
MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.
MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.
MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.
MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.
MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.
MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.
MQRC_TARGET_LENGTH_ERROR
(2144, X'860') Target length parameter not valid.
MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQGET (Hconn,Hobj,&MsgDesc,&GetMsgOpts,BufferLength,Buffer,
&DataLength,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQHOBJ Hobj;            /*Object handle */
MQMD MsgDesc;           /*Message descriptor */
MQGMO GetMsgOpts;       /*Options that control the action of MQGET */
MQLONG BufferLength;    /*Length in bytes of the Buffer area */
MQBYTE Buffer[n];       /*Area to contain the message data */
MQLONG DataLength;      /*Length of the message */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQGET'USING HCONN,HOBJ,MSGDESC,GETMSGOPTS,
BUFFERLENGTH,BUFFER,DATALENGTH,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object handle
  01 HOBJ PIC  S9(9)BINARY.
**Message descriptor
  01 MSGDESC.
      COPY CMQMDV.
**Options that control the action of MQGET
  01 GETMSGOPTS.
       COPY CMQGMOV.
**Length in bytes of the Buffer area
  01 BUFFERLENGTH PIC S9(9)BINARY.
**Area to contain the message data
  01 BUFFER PIC X(n).
**Length of the message
  01 DATALENGTH PIC S9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PIC S9(9)BINARY.
```

PL/I invocation

```
CALL MQGET (HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,BUFFER,
DATALENGTH,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31); /*Connection handle */
DCL HOBJ        FIXED BIN(31); /*Object handle */
DCL MSGDESC     LIKE MQMD;     /*Message descriptor */
DCL GETMSGOPTS  LIKE MQGMO;    /*Options that control the action of MQGET */
DCL BUFFERLENGTH FIXED BIN(31);/*Length in bytes of the Buffer area */
DCL BUFFER      CHAR(n);       /*Area to contain the message data */
DCL DATALENGTH  FIXED BIN(31); /*Length of the message */
DCL COMPCODE    FIXED BIN(31); /*Completion code */
DCL REASON      FIXED BIN(31); /*Reason code qualifying CompCode */
```

MQINQ - Inquire about object attributes

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object. The following types of object are valid:

- Queue
- Queue manager
- Namelist

Syntax

```
MQINQ (Hconn,Hobj,SelectorCount,Selectors,IntAttrCount,
IntAttrs,CharAttrLength,CharAttrs,CompCode,Reason)
```

Parameters

The MQINQ call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

Hobj (MQHOBJ) - input
Object handle.

This handle represents the object (of any type) whose attributes are required. The handle must have been returned by a previous MQOPEN call that specified the MQOO_INQUIRE option.

SelectorCount (MQLONG) - input
Count of selectors.

This is the count of selectors that are supplied in the Selectors array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

Selectors (MQLONG× SelectorCount) - input
Array of attribute selectors.

This is an array of SelectorCount attribute selectors; each selector identifies an attribute (integer or character) whose value is required. Each selector must be valid for the type of object that Hobj represents, otherwise the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

In the special case of queues:

- If the selector is not valid for queues of any type, the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

- If the selector is applicable only to queues of type or types other than that of the object, the call succeeds with completion code MQCC_WARNING and reason code MQRC_SELECTOR_NOT_FOR_TYPE.

Selectors for queue managers:

MQCA_ALTERATION_DATE
Last modification date (Length of 12, format 'YYYY-MM-DD ').

MQCA_ALTERATION_TIME
Last modification time (Length of 8, format 'HH:MM:SS').

MQCA_BATCH_INTERFACE_ID
Batch interface identifier (MQ_BATCH_INTERFACE_ID_LENGTH).

MQCA_CHANNEL_AUTO_DEF_EXIT
Automatic channel definition exit name (MQ_EXIT_NAME_LENGTH).

MQCA_COMMAND_INPUT_Q_NAME
System command queue name (MQ_Q_NAME_LENGTH).

MQCA_COMMAND_REPLY_Q_NAME
MQSC reply queue name (MQ_Q_NAME_LENGTH).

MQCA_DEAD_LETTER_Q_NAME
System dead letter queue name (MQ_Q_NAME_LENGTH).

MQCA_MONITOR_Q_NAME
MQI monitor queue name (MQ_Q_NAME_LENGTH).

MQCA_Q_MGR_DESC
Queue manager description (MQ_Q_MGR_DESC_LENGTH).

MQCA_Q_MGR_IDENTIFIER
Queue manager identifier (MQ_Q_MGR_IDENTIFIER_LENGTH).

MQCA_Q_MGR_NAME
Queue manager name (MQ_Q_MGR_NAME_LENGTH).

MQCA_SSL_KEY_LIBRARY
SSL key library name (MQ_SSL_KEY_LIBRARY_LENGTH).

MQCA_SSL_KEY_MEMBER
SSL key library member name (MQ_SSL_KEY_MEMBER_LENGTH).

MQCA_SYSTEM_LOG_Q_NAME
System log queue name (MQ_Q_NAME_LENGTH).

MQIA_ACCOUNTING_CONN_OVERRIDE
Accounting connection override setting.

MQIA_ACCOUNTING_INTERVAL
Accounting message interval.

MQIA_ACCOUNTING_MQI
MQI Accounting setting.

MQIA_ACCOUNTING_Q
Default queue accounting setting.

MQIA_ADOPTNEWMCA_CHECK
Indicates whether the Adopt MCA feature checks the partner net address when adopting an MCA instance. Can be one of the following values:
MQADOPT_CHECK_NONE
MQADOPT_CHECK_NET_ADDR

MQIA_ADOPTNEWMCA_TYPE
Indicates whether the channel Adopt MCA feature is active for the queue manager. Can be one of the following values:
MQADOPT_TYPE_NO
MQADOPT_TYPE_RCVR

MQIA_AUTHORITY_EVENT
Control attribute for authority events.

MQIA_BATCH_INTERFACE_AUTO
Indicator for the automatic activation of the batch interface. Can be one of the following values:
MQAUTO_START_NO
MQAUTO_START_YES

MQIA_CHANNEL_AUTO_DEF
Control attribute for automatic channel definition.

MQIA_CHANNEL_AUTO_DEF_EVENT
Control attribute for automatic channel definition events.

MQIA_CHANNEL_EVENT
Indicates whether channel-related events are generated. Can be one of the following values:
MQEVR_ENABLED
MQEVR_DISABLED

MQIA_CMD_SERVER_AUTO
Indicator for the automatic activation of the PCF command server. Can be one of the following values:
MQAUTO_START_NO
MQAUTO_START_YES

MQIA_CMD_SERVER_CONVERT_MSG
Indicator for the data conversion of PCF messages. Can be one of the following values:
MQCSRV_CONVERT_NO
MQCSRV_CONVERT_YES

MQIA_CMD_SERVER_DLQ_MSG
Indicator for the storage of undeliverable PCF reply messages. Can be one of the following values:
MQCSRV_DLQ_NO
MQCSRV_DLQ_YES

MQIA_CODED_CHAR_SET_ID
Local code page for queue manager.

MQIA_COMMAND_EVENT
Control attribute for command events. The value can be:
MQEVR_DISABLED
Event reporting disabled.
MQEVR_ENABLED
Event reporting enabled.
MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

MQIA_COMMAND_LEVEL
Supported command level. For WebSphere MQ for z/VSE, this value is always MQCMDL_LEVEL_211.

MQIA_CONFIGURATION_EVENT
Control attribute for configuration events. The value can be:
MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQIA_DIST_LISTS
Indicator for distributed list support. Can be one of the following values:
MQDL_SUPPORTED
MQDL_NOT_SUPPORTED

MQIA_INHIBIT_EVENT
Control attribute for inhibit events.

MQIA_LISTENER_PORT_NUMBER
Port number for TCP/IP Listener process.

MQIA_LOCAL_EVENT
Control attribute for local events.

MQIA_MAX_CLIENTS
Specifies the maximum number of licensed clients that can establish a server connection at any one time.

MQIA_MAX_GLOBAL_LOCKS
Buffer size for queue manager to manage concurrent queue access.

MQIA_MAX_HANDLES
Maximum number of concurrent connections to the queue manager.

MQIA_MAX_LOCAL_LOCKS
Buffer size for applications to manage concurrent queue access.

MQIA_MAX_MSG_LENGTH
Maximum message length for queue messages.

MQIA_MAX_OPEN_Q
Maximum number of concurrently open queues.

MQIA_MAX_Q_DEPTH
Maximum allowable queue depth for queues.

MQIA_MAX_RECOVERY_TASKS
Indicates the maximum number of CICS tasks that the queue manager will start to resolve discrepancies in dual queues.

MQIA_MONITOR_INTERVAL
Queue manager housekeeping process interval.

MQIA_MONITORING_CHANNEL
Default channel monitoring setting.

MQIA_MONITORING_Q
Default queue monitoring setting.

MQIA_PERFORMANCE_EVENT
Control attribute for performance events.

MQIA_PLATFORM
WebSphere MQ system platform identifier. For WebSphere MQ for z/VSE, this value is always MQPL_VSE.

MQIA_Q_USERS
Maximum number of concurrent opens per queue.

MQIA_QMOPT_CONS_COMMS_MSGS
Indicates whether the queue manager sends communication related messages to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED
MQQMOPT_REPLY

MQIA_QMOPT_CONS_CRITICAL_MSGS
Indicates whether the queue manager sends messages of critical severity to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED
MQQMOPT_REPLY

MQIA_QMOPT_CONS_ERROR_MSGS
Indicates whether the queue manager sends messages of error severity to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED
MQQMOPT_REPLY

MQIA_QMOPT_CONS_INFO_MSGS
Indicates whether the queue manager sends messages of informational severity to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_CONS_REORG_MSGS
Indicates whether the queue manager sends reorganization related messages to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED
MQQMOPT_REPLY

MQIA_QMOPT_CONS_SYSTEM_MSGS
Indicates whether the queue manager sends general system related messages to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED
MQQMOPT_REPLY

MQIA_QMOPT_CONS_WARNING_MSGS
Indicates whether the queue manager sends messages of warning severity to the console. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_CSMT_ON_ERROR
Indicates whether operational messages are sent to the CICS CSMT when the system log queue is unavailable. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_INTERNAL_DUMP
Indicates whether the queue manager generates a CICS dump when an MQI application generates an unrecoverable error. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_COMMS_MSGS
Indicates whether the queue manager sends communication related messages to the system log. Can be one of the following values:

MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_CRITICAL_MSGS
Indicates whether the queue manager sends messages of critical severity to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_ERROR_MSGS
Indicates whether the queue manager sends messages of error severity to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_INFO_MSGS
Indicates whether the queue manager sends messages of informational severity to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_REORG_MSGS
Indicates whether the queue manager sends reorganization related messages to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_SYSTEM_MSGS
Indicates whether the queue manager sends general system related messages to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_LOG_WARNING_MSGS
Indicates whether the queue manager sends messages of warning severity to the system log. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_TRACE_COMMS
Indicates whether the queue manager traces communication related events. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_TRACE_CONVERSION
Indicates whether the queue manager traces data conversion related events. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_TRACE_MQI_CALLS
Indicates whether the queue manager traces MQI call related events. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_TRACE_REORG
Indicates whether the queue manager traces reorganization related events. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_QMOPT_TRACE_SYSTEM
Indicates whether the queue manager traces general system related events. Can be one of the following values:
MQQMOPT_ENABLED
MQQMOPT_DISABLED

MQIA_REMOTE_EVENT
Control attribute for remote events.

MQIA_SSL_EVENT
Control attribute for channel events. The value can be:
MQEVR_DISABLED
Event reporting disabled.
MQEVR_ENABLED
Event reporting enabled.

MQIA_SSL_RESET_COUNT
SSL key reset count.

MQIA_START_STOP_EVENT
Control attribute for start stop events.

MQIA_STATISTICS_CHANNEL
Default channel statistics setting.

MQIA_STATISTICS_INTERVAL
Statistics message interval.

MQIA_STATISTICS_MQI
MQI Statistics setting.

MQIA_STATISTICS_Q
Default queue statistics setting.

MQIA_SYNCPOINT
Indicator for SYNCPOINT support. For WebSphere MQ for z/VSE, this value is always MQSP_NOT_AVAILABLE.

Selectors for all types of queue:

MQCA_CREATION_DATE
Queue creation date (Length of 12, format 'YYYY-MM-DD ').

MQCA_CREATION_TIME
Queue creation time (Length of 8, format 'HH:MM:DD').

MQCA_Q_DESC
Queue description (MQ_Q_DESC_LENGTH).

MQCA_Q_NAME
Queue name (MQ_Q_NAME_LENGTH).

MQIA_INHIBIT_PUT
Whether put operations are allowed. Can be one of the following values:
MQQA_PUT_ALLOWED
MQQA_PUT_INHIBITED

MQIA_Q_TYPE
Queue type. Can be one of the following values:
MQQT_ALIAS
MQQT_LOCAL
MQQT_REMOTE

Selectors for local queues:

MQCA_AUTO_REORG_CATALOG
The contents of this field are now treated as comments and may not reflect the actual VSAM catalog containing the reorganization file. For the reorganization process, the VSAM catalog where the reorganization file is defined is now extracted from the system and so no longer needs to be specified in the queue definition.

MQCA_AUTO_REORG_START_TIME
Indicates the time of day, in HHMM format, for the automatic VSAM reorganization to occur following a system restart (MQ_AUTO_REORG_TIME_LENGTH).

MQCA_CICS_FILE_NAME
CSD file name for queue messages (MQ_CICS_FILE_NAME_LENGTH).

MQCA_TRIGGER_CHANNEL_NAME
Channel name for MCA trigger process (MQ_CHANNEL_NAME_LENGTH).

MQCA_TRIGGER_DATA
Trigger user data (MQ_PROCESS_USER_DATA_LENGTH).

MQCA_TRIGGER_PROGRAM_NAME
Program name for trigger process (MQ_TRIGGER_PROGRAM_NAME_LENGTH).

MQCA_TRIGGER_TERM_ID
Terminal identifier for trigger process (MQ_TRIGGER_TERM_ID_LENGTH).

MQCA_TRIGGER_TRANS_ID
Transaction identifier for trigger process (MQ_TRIGGER_TRANS_ID_LENGTH).

MQIA_ACCOUNTING_Q
Queue accounting setting.

MQIA_AUTO_REORG_INTERVAL
Indicates the frequency, in minutes, for the automatic VSAM reorganization to occur, after its initial activation.

MQIA_AUTO_REORGANIZATION
Indicates whether the VSAM file hosting a queue should be scheduled for automatic VSAM reorganization. Can be one of the following values:
MQREORG_ENABLED
MQREORG_DISABLED

MQIA_CURRENT_Q_DEPTH
Current queue depth.

MQIA_DEF_PERSISTENCE
Default persistence for queue. For WebSphere MQ for z/VSE, this value is always MQPER_PERSISTENT.

MQIA_DEFINITION_TYPE
Queue definition type. For WebSphere MQ for z/VSE, this value is always MQQDT_PREDEFINED.

MQIA_INHIBIT_GET
Whether get operations are allowed. Can be one of the following values:
MQQA_GET_ALLOWED
MQQA_GET_INHIBITED

MQIA_MAX_GLOBAL_LOCKS
Buffer size for queue manager to manage concurrent queue access.

MQIA_MAX_LOCAL_LOCKS
Buffer size for applications to manage concurrent queue access.
MQIA_MAX_MSG_LENGTH
Maximum message length for queue messages.
MQIA_MAX_Q_DEPTH
Maximum allowable queue depth for queues.
MQIA_MAX_Q_TRIGGERS
Maximum number of concurrent trigger instances for a particular queue.
MQIA_MONITORING_Q
Queue monitoring setting.
MQIA_OPEN_INPUT_COUNT
Number of opens for input currently issued against queue.
MQIA_OPEN_OUTPUT_COUNT
Number of opens for output currently issued against queue.
MQIA_Q_DEPTH_HIGH_EVENT
Control attribute for queue depth high events.
MQIA_Q_DEPTH_HIGH_LIMIT
High limit for queue depth.
MQIA_Q_DEPTH_LOW_EVENT
Control attribute for queue depth low events.
MQIA_Q_DEPTH_LOW_LIMIT
Low limit for queue depth.
MQIA_Q_DEPTH_MAX_EVENT
Control attribute for queue depth max events.
MQIA_Q_SERVICE_INTERVAL
Limit for queue service interval.
MQIA_Q_SERVICE_INTERVAL_EVENT
Control attribute for queue service interval events.
MQIA_Q_USERS
Maximum number of concurrent opens per queue.
MQIA_SHAREABILITY
Queue shareability mode. Can be one of the following values:
MQQA_SHAREABLE
MQQA_NOT_SHAREABLE
MQIA_STATISTICS_Q
Queue statistics setting.
MQIA_TRIGGER_CONTROL
Whether a trigger is required for the queue. Can be one of the following values:
MQTC_OFF
MQTC_ON
MQIA_TRIGGER_RESTART
Indicator for the reactivation of a trigger process. Can be one of the following values:
MQTRIGGER_RESTART_YES
MQTRIGGER_RESTART_NO
MQIA_TRIGGER_TYPE
Trigger event type. Can be one of the following values:
MQTT_NONE
MQTT_FIRST
MQTT_EVERY

MQIA_USAGE
Queue usage. Can be one of the following values:
MQUS_NORMAL
MQUS_TRANSMISSION

Selectors for local definitions of remote queues

MQCA_REMOTE_Q_MGR_NAME
Name of remote queue manager (MQ_Q_MGR_NAME_LENGTH).

MQCA_REMOTE_Q_NAME
Name of remote queue as known on remote queue manager (MQ_Q_NAME_LENGTH).

MQCA_XMIT_Q_NAME
Name of local transmission queue.

Selectors for alias queues

MQCA_BASE_Q_NAME
Name of queue that alias resolves to (MQ_Q_NAME_LENGTH).

MQIA_INHIBIT_GET
Whether get operations are allowed.

Selectors for namelists

MQCA_ALTERATION_DATE
Date of most-recent alteration (MQ_DATE_LENGTH).

MQCA_ALTERATION_TIME
Time of most-recent alteration (MQ_TIME_LENGTH).

MQCA_NAMELIST_NAME
Name of namelist object (MQ_NAMELIST_NAME_LENGTH).

MQCA_NAMELIST_DESC
Description of namelist object (MQ_NAMELIST_DESC_LENGTH).

MQCA_NAMES
Names in the namelist. Namelist names have a length specified by the MQ_OBJECT_NAME_LENGTH constant.

MQIA_NAME_COUNT
Names of names in the namelist.

IntAttrCount (MQLONG) - input
Count of integer attributes.

This is the number of elements in the IntAttrs array. Zero is a valid value. If this is at least the number of MQIA_* selectors in the Selectors parameter, all integer attributes requested are returned.

IntAttrs (MQLONG×IntAttrCount) - output
Array of integer attributes.

This is an array of IntAttrCount integer attribute values.

Integer attribute values are returned in the same order as the MQIA_* selectors in the Selectors parameter. If the array contains more elements than the number of MQIA_* selectors, the excess elements are unchanged.

If Hobj represents a queue, but an attribute selector is not applicable to that type of queue, the specific value MQIAV_NOT_APPLICABLE is returned for the corresponding element in the IntAttrs array.

If the IntAttrCount or SelectorCount parameter is zero, IntAttrs is not referred to; in this case, the parameter address passed by programs written in C or S/390 assembler may be null.

CharAttrLength (MQLONG) - input
Length of character attributes buffer.

This is the length in bytes of the CharAttrs parameter.

This must be at least the sum of the lengths of the requested character attributes (see Selectors). Zero is a valid value.

CharAttrs (MQCHAR×CharAttrLength) - output
Character attributes.

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the CharAttrLength parameter. Character attributes are returned in the same order as the MQCA_* selectors in the Selectors parameter. The length of each attribute string is fixed for each attribute (see Selectors), and the value in it is padded to the right with blanks if necessary. If the buffer is larger than that needed to contain all of the requested character attributes (including padding), the bytes beyond the last attribute value returned are unchanged.

If Hobj represents a queue, but an attribute selector is not applicable to that type of queue, a character string consisting entirely of asterisks (*) is returned as the value of that attribute in CharAttrs.

If the CharAttrLength or SelectorCount parameter is zero, CharAttrs is not referred to; in this case, the parameter address passed by programs written in C may be null.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:
MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.
MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.
MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.
MQRC_NOT_OPEN_FOR_INQUIRE
(2038, X'7F6') Queue not open for inquire.
MQRC_Q_DELETED
(2052, X'804') The queue has been deleted.
MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.
MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.
MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too big.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQINQ (Hconn,Hobj,SelectorCount,Selectors,IntAttrCount,IntAttrs,
CharAttrLength,CharAttrs,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQHOBJ Hobj;            /*Object handle */
MQLONG SelectorCount;   /*Count of selectors */
MQLONG Selectors [n];   /*Array of attribute selectors */
MQLONG IntAttrCount;    /*Count of integer attributes */
MQLONG IntAttrs [n];    /*Array of integer attributes */
MQLONG CharAttrLength;  /*Length of character attributes buffer */
MQCHAR CharAttrs [n];   /*Character attributes */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQ'USING HCONN,HOBJ,SELECTORCOUNT, SELECTORS-
TABLE,INTATTRCOUNT,INTATTRS-TABLE,
CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object handle
  01 HOBJ PICS9(9)BINARY.
**Count of selectors
  01 SELECTORCOUNT PIC S9(9)BINARY.
**Array of attribute selectors
  01 SELECTORS-TABLE.
     02 SELECTORS PIC S9(9)BINARY OCCURS n TIMES.
**Count of integer attributes
  01 INTATTRCOUNT PIC S9(9)BINARY.
**Array of integer attributes
  01 INTATTRS-TABLE.
     02 INTATTRS PIC S9(9)BINARY OCCURS n TIMES.
**Length of character attributes buffer
  01 CHARATTRLENGTH PIC S9(9)BINARY.
**Character attributes
  01 CHARATTRS PIC X(n).
```

```
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQINQ (HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,
INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN FIXED    BIN(31);        /*Connection handle */
DCL HOBJ FIXED     BIN(31);        /*Object handle */
DCL SELECTORCOUNT  FIXED BIN(31);  /*Count of selectors */
DCL SELECTORS(N)   FIXED BIN(31);  /*Array of attribute selectors */
DCL INTATTRCOUNT   FIXED BIN(31);  /*Count of integer attributes */
DCL INTATTRS(N)    FIXED BIN(31);  /*Array of integer attributes */
DCL CHARATTRLENGTH FIXED BIN(31);  /*Length of character attributes buffer */
DCL CHARATTRS      CHAR(N);        /*Character attributes */
DCL COMPCODE       FIXED BIN(31);  /*Completion code */
DCL REASON         FIXED BIN(31);  /*Reason code qualifying CompCode */
```

MQINQMP - Inquire message property

The MQINQMP call returns the value of a property of a message.

Syntax

```
MQINQMP ( Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type, ValueLength,
Value, DataLength, CompCode,Reason )
```

Parameters

The MQINQMP call has the following parameters:

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value of Hconn must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input

This is the message handle to be inquired. The value was returned by a previous MQCRTMH call.

InqPropOpts (MQIMPO) - Input

See the MQIMPO data type for details.

Name (MQCHARV) - input

The name of the property to inquire.

If no property with this name can be found, the call fails with reason MQRC_PROPERTY_NOT_AVAILABLE.

You can use the wildcard character "%" at the end of the property name. The wildcard matches zero or more characters, including the "." character. This allows an application to inquire the value of many properties.

Call MQINQMP with option MQIMPO_INQ_FIRST to get the first matching property and again with the option MQIMPO_INQ_NEXT to get the next matching property. When no more matching properties are available, the call fails with MQRC_PROPERTY_NOT_AVAILABLE.

If the ReturnedName field of the InqPropOpts structure is initialized with an address or offset for the returned name of the property, this is filled in on return from MQINQMP with the same address or offset of the property that has been matched. If the VSBufSize field of the ReturnedName in the InqPropOpts structure is less than the length of the returned property name, the completion code is set MQCC_FAILED with reason MQRC_PROPERTY_NAME_TOO_BIG.

Properties that have known synonyms are returned as follows:

- Properties with the prefix "mqps." are returned with the MQ property name. For example, "MQTopicString" is the returned name rather than "mqps.Top".
- Properties with the prefix "jms." or "mcd." are returned as the JMS header field name. For example, "JMSExpiration" is the returned name rather than "jms.Exp".
- Properties with the prefix "usr." are returned without that prefix. For example, "Color" is returned rather than "usr.Color".

Properties with synonyms are only returned once.

In the C programming language, the following macro variables are defined for inquiring on all properties and all properties that begin "usr" respectively:

MQPROP_INQUIRE_ALL
Inquire on all properties of the message.

MQPRP_INQUIRE_ALL_USR
Inquire on all properties of the message that start "usr.". The returned name is returned without the "usr." prefix.

If MQIMP_INQ_NEXT is specified but Name has changed since the previous call, or this is the first call, then MQIMPO_INQ_FIRST is implied.

PropDesc (MQPD) - output
This structure is used to define the attributes of a property, including what happens if the property is not supported, what message context the property belongs to, and what messages the property should be copied into. See MQPD for details of this structure.

Type (MQLONG) - input/output
On return from the MQINQMP call, this parameter is set to the data type of Value. The data type can be any of the following:

MQTYPE_BOOLEAN
A boolean.

MQTYPE_BYTE_STRING
A byte string.

MQTYPE_INT8
An 8-bit signed integer.

MQTYPE_INT16
A 16-bit signed integer.

MQTYPE_INT32
A 32-bit signed integer.

MQTYPE_FLOAT32
A 32-bit floating-point number.

MQTYPE_FLOAT64
A 64-bit floating-point number.

MQTYPE_STRING
A character string.

MQTYPE_NULL
The property exists but has a null value.

If the data type of the property value is not recognized, then MQTYPE_STRING is returned and a string representation of the value is placed into the Value area. A string representation of the data type can be found in the TypeString field of the InqPropOpts parameter. A warning completion code is returned with reason MQRC_PROP_TYPE_NOT_SUPPORTED.

Additionally, if the option MQIMPO_CONVERT_TYPE is specified, conversion of the property value is requested. Use Type as an input to specify the data type that you want the property to be returned as. See the description of the MQIMPO_CONVERT_TYPE option of the MQIMPO structure for details of data type conversion.

If you do not request type conversion, you can use the following value on input:

MQTYPE_AS_SET
The value of the property is returned without converting its data type.

ValueLength (MQLONG) - input

The length in bytes of the Value area. Specify zero for properties that you do not require the value returned for. These could be properties which are designed by an application to have a null value or an empty string. Also specify zero if the MQIMPO_QUERY_LENGTH option has been specified; in this case no value is returned.

Value (MQBYTExValueLength) - output
This is the area to contain the inquired property value. The buffer should be aligned on a boundary appropriate for the value being returned. Failure to do so may result in an error when the value is later accessed.

If ValueLength is less than the length of the property value, as much of the property value as possible is moved into Value and the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_VALUE_TOO_BIG.

The character set of the data in Value is given by the ReturnedCCSID field in the InqPropOpts parameter. The encoding of the data in Value is given by the ReturnedEncoding field in the InqPropOpts parameter.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the ValueLength parameter is zero, Value is not referred to and its value passed by programs written in C can be null.

DataLength (MQLONG) - output

This is the length in bytes of the actual property value as returned in the Value area.

If DataLength is less than the property value length, DataLength is still filled in on return from the MQINQMP call. This allows the application to determine the size of the buffer required to accommodate the property value, and then reissue the call with a buffer of the appropriate size.

The following values may also be returned.

If the Type parameter is set to MQTYPE_STRING or MQTYPE_BYTE_STRING:

MQVL_EMPTY_STRING
The property exists but contains no characters or bytes.

CompCode (MQLONG) - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) - output

The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_WARNING:

MQRC_PROP_NAME_NOT_CONVERTED
(2492, X'09BC') Returned property name not converted.

MQRC_PROP_VALUE_NOT_CONVERTED
(2466, X'09A2') Property value not converted.

MQRC_PROP_TYPE_NOT_SUPPORTED
(2467, X'09A3') Property data type is not supported.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If CompCode is MQCC_FAILED:

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_IMPO_ERROR
(2464, X'09A0') Inquire message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'07F8') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROP_CONV_NOT_SUPPORTED
(2470, X'09A6') Conversion from the actual to requested data type not supported.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_NAME _TOO_BIG
(2465, X'09A1') Property name too big for returned name buffer.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') Property value too big for the Value area.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid requested property type.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'0871') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQINQMP (  Hconn, Hmsg, &InqPropOpts, &Name, &PropDesc,
&Type,ValueLength, Value, &DataLength, &CompCode, &Reason);
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQDIMPO InqPropOpts; /* Options that control the action of MQINQMP */
MQCHARV Name; /* Property name */
MQPD PropDesc; /* Property descriptor */
MQLONG Type; /* Property data type */
MQLONG ValueLength; /* Length in bytes of the Value area */
MQBYTE Value[n]; /* Area to contain the property value */
MQLONG DataLength; /* Length of the property value */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQMP' USING HCONN, HMSG, INQMSGOPTS, NAME, PROPDESC,
TYPE,VALUELENGTH, VALUE, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQINQMP
01 INQMSGOPTS.
   COPY CMQIMPOV.
** Property name
01 NAME.
   COPY CMQCHRVV.
** Property descriptor
01 PROPDESC.
   COPY CMQPDV.
** Property data type
01 TYPE PIC S9(9) BINARY.
** Length in bytes of the VALUE area
01 VALUELENGTH PIC S9(9) BINARY.
** Area to contain the property value
01 VALUE PIC X(n).
** Length of the property value
01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQMP ( Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type,
ValueLength, Value, DataLength, CompCode, Reason );
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8); /* Message handle */
dcl InqPropOpts like MQIMPO;
                 /* Options that control the action of MQINQMP */
dcl Name like MQCHARV; /* Property name */
dcl PropDesc like MQPD; /* Property descriptor */
dcl Type fixed bin (31); /* Property data type */
dcl ValueLength fixed bin (31); /* Length in bytes of the Value area */
dcl Value char (n); /* Area to contain the property value */
dcl DataLength fixed bin (31); /* Length of the property value */
dcl CompCode fixed bin (31); /* Completion code */
dcl Reason fixed bin (31); /* Reason code qualifying CompCode */
```

MQMHBUF - Convert message handle into buffer

The MQMHBUF converts a message handle into a buffer and is the inverse of the MQBUFMH call.

Syntax

```
MQMHBUF ( Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer,
DataLength, CompCode, Reason )
```

Parameters

The MQMHBUF call has the following parameters:

Hconn (MQHCONN) - input

This handle represents the connection to the queue manager. The value of Hconn must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input

This is the message handle for which a buffer is required.

The value was returned by a previous MQCRTMH call.

MsgHBufOpts (MQMHBO) - input

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles.

See "MQMHBO – Message handle to buffer options" on page 810 for details.

Name (MQCHARV) - input

The name of the property or properties to put into the buffer.

If no property matching the name can be found, the call fails with MQRC_PROPERTY_NOT_AVAILABLE.

Wildcards:

You can use a wildcard to put more than one property into the buffer. To do this, use the wildcard character "%" at the end of the property name. This wildcard matches zero or more characters, including the "." character.

In the C programming language, these macro variables are defined for inquiring on all properties and all properties that begin "usr", respectively:

MQPROP_INQUIRE_ALL
Put all properties of the message into the buffer.

MQPROP_INQUIRE_ALL_USR
Put all properties of the message that start with the characters "usr." into the buffer.

MsgDesc (MQMD) - input/output

The MsgDesc structure describes the contents of the buffer area.

On output, the Encoding, CodedCharSetId, and Format fields are set to correctly describe the encoding, character set identifier and format of the data in the buffer area as written by the call.

Data in this structure is in the character set and encoding of the application.

BufferLength (MQLONG) - input

BufferLength is the length of the Buffer area, in bytes.

Buffer (MQBYTExBufferLength) - output

Buffer defines the area to contain the message properties. You should align the buffer on a 4-byte boundary.

If BufferLength is less than the length required to store the properties in Buffer, MQMHBUF fails with MQRC_PROPERTY_VALUE_TOO_BIG. The contents of the buffer can change even if the call fails.

DataLength (MQLONG) - output

DataLength is the length, in bytes, of the returned properties in the buffer. If the value is zero, no properties matched the value given in Name and the call fails with reason code MQRC_PROPERTY_NOT_AVAILABLE.

If BufferLength is less than the length required to store the properties in the buffer, the MQMHBUF call fails with

MQRC_PROPERTY_VALUE_TOO_BIG, but a value is still entered into DataLength. This allows the application to determine the size of the buffer required to accommodate the properties, and then reissue the call with the required BufferLength.

CompCode (MQLONG) - output

The completion code; it is has one of these values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output

The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:

MQRC_MHBO_ERROR
(2501, X'095C') Message handle to buffer options structure not valid.

MQRC_BUFFER_ERROR
(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Buffer length parameter not valid.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Property name is not valid.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') BufferLength value is too small to contain specified properties.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Usage notes

MQMHBUF converts a message handle into a buffer.

You can use it with an MQGET API exit to access certain properties, using the message property APIs, and then pass these in a buffer back to an application designed to use MQRFH2 headers rather than message handles.

This call is the inverse of the MQBUFMH call, which you can use to parse message properties from a buffer into a message handle.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQMHBUF ( Hconn, Hmsg, &MsgHBufOpts, &Name, &MsgDesc, BufferLength,
Buffer, &DataLength, &CompCode, &Reason );
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQMHBO MsgHBufOpts; /* Options that control the action of MQMHBUF */
MQCHARV Name; /* Property name */
MQMD MsgDesc; /* Message descriptor */
MQLONG BufferLength; /* Length in bytes of the Buffer area */
MQBYTE Buffer[n]; /* Area to contain the properties */
MQLONG DataLength; /* Length of the properties */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQMHBUF' USING   HCONN, HMSG, MSGHBUFOPTS, NAME, MSGDESC,
BUFFERLENGTH, BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQMHBUF
01 MSGHBUFOPTS.
COPY CMQMHBOV.
** Property name
01 NAME
   COPY CMQCHRVV.
** Message descriptor
01 MSGDESC
   COPY CMQMDV.
** Length in bytes of the Buffer area */
01 BUFFERLENGTH PIC S9(9) BINARY.
** Area to contain the properties
01 BUFFER PIC X(n).
** Length of the properties
01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQMHBUF ( Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc,
BufferLength, Buffer,DataLength, CompCode, Reason );
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8); /* Message handle */
dcl MsgHBufOpts like MQMHBO;
                 /* Options that control the action of MQMHBUF */
dcl Name like MQCHARV; /* Property name */
dcl MsgDesc like MQMD; /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer char(n); /* Area to contain the properties */
dcl DataLength fixed bin(31); /* Length of the properties */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQOPEN - Open object

The MQOPEN call establishes access to an object. The following types of object are valid:

Queue
Queue manager
Namelist

Syntax

```
MQOPEN (Hconn,ObjDesc,Options,Hobj,CompCode,Reason)
```

Parameters

The MQOPEN call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

ObjDesc (MQOD) - input/output
Object descriptor.

This is a structure that identifies the object to be opened; see "MQOD – Object descriptor" on page 813 for details.

Options (MQLONG) - input
Options that control the action of MQOPEN.

The following options apply and you must specify at least one of these. However, you cannot specify the two input options together and you cannot specify an input option with an output option.

MQOO_BROWSE
MQOO_INPUT_SHARED
MQOO_INPUT_EXCLUSIVE
MQOO_INQUIRE
MQOO_OUTPUT
MQOO_SET

Note that namelist objects can only be opened with the MQOO_INQUIRE option.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another

application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

This option is valid only for local, alias, and model queues; it is not valid for remote queues.

The following notes apply to these options:

- Only one of these options can be specified.
- An MQOPEN call with one of these options can succeed even if the InhibitGet queue attribute is set to MQQA_GET_INHIBITED (although subsequent MQGET calls will fail while the attribute is set to this value).
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if ObjectQMgrName is the name of a queue manager alias; this is true even if the value of the RemoteQMgrName attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_BROWSE
Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

1. MQGMO_BROWSE_FIRST
2. MQGMO_BROWSE_NEXT
3. MQGMO_BROWSE_MSG_UNDER_CURSOR

This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue; see "MQGET - Get message" on page 888 for further information.

This option is valid only for local and alias; it is not valid for remote queues and objects which are not queues. It is also not valid if ObjectQMgrName is the name of a queue manager alias; this is true even if the value of the RemoteQMgrName attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_OUTPUT
Open queue to put messages.

The queue is opened for use with subsequent MQPUT calls. An MQOPEN call with this option can succeed even if the InhibitPut queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls will fail while the attribute is set to this value).

MQOO_INQUIRE
Open object to inquire attributes.

The queue or queue manager is opened for use with subsequent MQINQ calls.

The queue, queue manager or namelist is opened for use with subsequent MQINQ calls.

MQOO_SET
Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls. This option is valid for all queue types supported by WebSphere MQ for z/VSE.

Hobj (MQHOBJ) - output
Object handle.

This handle represents the access that has been established to the object. It must be specified on subsequent message queuing calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the CICS task terminates.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_FAILED:
MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_DYNAMIC_Q_NAME_ERROR
(2011, X'7DB') The dynamic queue name is invalid.
MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_OBJECT_ALREADY_EXISTS
(2100, X'834') For a dynamic queue, the queue name already exists.
MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.
MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.
MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.
MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') Option not valid for object type.
MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.
MQRC_Q_DELETED
(2052, X'804') The queue has been deleted.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.
MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.
MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.
MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.
MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

Usage notes

1. The object opened is one of the following:
   - A queue, in order to:
     - Get or browse messages (using the MQGET call).
     - Put messages (using the MQPUT call).
     - Inquire about the attributes of the queue (using the MQINQ call).
     - Set the attributes of the queue (using the MQSET call).
   - The queue manager, in order to:
     - Inquire about the attributes of the local queue manager (using the MQINQ call).
   - A namelist, in order to:
     - Inquire about the attributes of the namelist object (using the MQINQ call).
2. It is valid for an application to open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.
3. If security is enabled, the queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. 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.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQOPEN (Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQOD ObjDesc;           /*Object descriptor */
MQLONG Options;         /*Options that control the action of MQOPEN */
MQHOBJ Hobj;            /*Object handle */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQOPEN'USING HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object descriptor
  01 OBJDESC.
  COPY CMQODV.
**Options that control the action of MQOPEN
  01 OPTIONS PICS9(9)BINARY.
**Object handle
  01 HOBJ PICS9(9)BINARY.
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQOPEN (HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN       FIXED BIN(31); /*Connection handle */
DCL OBJDESC     LIKE MQOD;     /*Object descriptor */
DCL OPTIONS     FIXED BIN(31); /*Options that control the action of
                                  MQOPEN */
DCL HOBJ        FIXED BIN(31); /*Object handle */
DCL COMPCODE    FIXED BIN(31); /*Completion code */
DCL REASON      FIXED BIN(31); /*Reason code qualifying CompCode */
```

MQPUT - Put message

The MQPUT call puts a message on a queue. The queue must already be open.

Syntax

```
MQPUT (Hconn,Hobj,MsgDesc,PutMsgOpts,BufferLength, Buffer,CompCode,Reason)
```

Parameters

The MQPUT call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

Hobj (MQHOBJ) - input
Object handle.

This handle represents the queue to which the message is added. The value of Hobj was returned by a previous MQOPEN call that specified the MQOO_OUTPUT option.

This handle can also represent an open distribution list.

MsgDesc (MQMD) - input/output
Message descriptor.

This structure describes the attributes of the message being sent, and receives information about the message after the put request is complete. See “MQMD – Message descriptor” on page 774 for details.

PutMsgOpts (MQPMO) - input/output
Put message options.

Options that control the action of MQPUT. See “MQPMO – Put message options” on page 826 for details.

BufferLength (MQLONG) - input
Length of the message in Buffer.

Zero is valid, and indicates that the message contains no application data.

Buffer (MQBYTE×BufferLength) - input
Message data.

This is a buffer containing the application data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment.

If Buffer contains character and/or numeric data, the CodedCharSetId and Encoding fields in the MsgDesc parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the CodedCharSetId queue-manager attribute and MQENC_NATIVE, respectively).

In the C programming language, the parameter is declared as a pointer-to-void; this means that the address of any type of data can be specified as the parameter. If the BufferLength parameter is zero, Buffer is not referred to; in this case, the parameter address passed by programs written in C can be null.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_WARNING
Warning (partial completion).
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_WARNING:
MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

If CompCode is MQCC_FAILED:
MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.
MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.
MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue.
MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.
MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.
MQRC_NOT_OPEN_FOR_OUTPUT
(2039, X'7F7') Queue not open for output.
MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.
MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.
MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Persistence specified is inconsistent with the queue.
MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.
MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.
MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.
MQRC_Q_DELETED
(2052, X'804') The queue has been deleted.
MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.
MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.
MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.
MQRC_UNKNOWN_CCSID
(2115, X'843') Unknown CCSID.

Language invocations

This call is supported in the following programming languages:
C invocation

```
MQPUT (Hconn,Hobj,&MsgDesc,&PutMsgOpts,BufferLength,Buffer,
&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQHOBJ  Hobj;           /*Object handle */
MQMD    MsgDesc;        /*Message descriptor */
MQPMO   PutMsgOpts;     /*Options that control the action of MQPUT */
MQLONG  BufferLength;   /*Length of the message in Buffer */
```

```
MQBYTE  Buffer [n ];    /*Message data */
MQLONG  CompCode;       /*Completion code */
MQLONG  Reason;         /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQPUT'USING HCONN,HOBJ,MSGDESC,PUTMSGOPTS,
BUFFERLENGTH,BUFFER,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object handle
  01 HOBJ PIC S9(9)BINARY.
**Message descriptor
  01 MSGDESC.
  COPY CMQMDV.
**Options that control the action of MQPUT
  01 PUTMSGOPTS.
  COPY CMQPMOV.
**Length of the message in Buffer
  01 BUFFERLENGTH PIC S9(9)BINARY.
**Message data
  01 BUFFER PIC X(n).
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQPUT (HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,BUFFER,
COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN        FIXED BIN(31); /*Connection handle */
DCL HOBJ         FIXED BIN(31); /*Object handle */
DCL MSGDESC      LIKE MQMD;     /*Message descriptor */
DCL PUTMSGOPTS   LIKE MQPMO;    /*Options that control the action of
                                    MQPUT */
DCL BUFFERLENGTH FIXED BIN(31); /*Length of the message in Buffer */
DCL BUFFER       CHAR(N);       /*Message data */
DCL COMPCODE     FIXED BIN(31); /*Completion code */
DCL REASON       FIXED BIN(31); /*Reason code qualifying CompCode */
```

MQPUT1 - Put one message

The MQPUT1 call puts one message on a queue. The queue need not be open.

Note: You cannot issue an MQPUT1 call to a model queue.

Syntax

```
MQPUT1 (Hconn,ObjDesc,MsgDesc,PutMsgOpts,BufferLength,
Buffer,CompCode,Reason)
```

Parameters

The MQPUT1 call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

ObjDesc (MQOD) - input/output
Object descriptor.

This is a structure which identifies the queue to which the message is added. See "MQOD – Object descriptor" on page 813 for details.

If security is enabled, the user must be authorized to open the queue for output.

MsgDesc (MQMD) - input/output
Message descriptor.

This structure describes the attributes of the message being sent, and receives feedback information after the put request is complete. See "MQMD – Message descriptor" on page 774 for details.

PutMsgOpts (MQPMO) - input/output
Put message options.

Options that control the action of MQPUT1. See "MQPMO – Put message options" on page 826 for details.

BufferLength (MQLONG) - input
Length of the message in Buffer.

Zero is valid, and indicates that the message contains no application data.

Buffer (MQBYTE×BufferLength) - input
Message data.

This is a buffer containing the application data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment.

If Buffer contains character and/or numeric data, the CodedCharSetId and Encoding fields in the MsgDesc parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the CodedCharSetId queue-manager attribute and MQENC_NATIVE, respectively).

In the C programming language, the parameter is declared as a pointer-to-void; this means that the address of any type of data can be specified as the parameter. If the BufferLength parameter is zero, Buffer is not referred to; in this case, the parameter address passed by programs written in C can be null.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_WARNING
Warning (partial completion).
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_WARNING:
MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

If CompCode is MQCC_FAILED:
MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.
MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.
MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.
MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.
MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue.
MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.
MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.
MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.
MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.
MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.
MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.
MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Persistence specified is inconsistent with the queue.
MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.
MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.
MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.
MQRC_Q_DELETED
(2052, X'804') The queue has been deleted.
MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.
MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.
MQRC_Q_TYPE_ERROR
(2057, X'809') Cannot MQPUT1 to a model queue.
MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.
MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.
MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.
MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.
MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.
MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

Usage notes

Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:

- The MQPUT call should be used when multiple messages are to be placed on the same queue.

  An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
- The MQPUT1 call should be used when only one message is to be put on a queue. This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, thereby minimizing the number of calls that must be issued.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQPUT1 (Hconn,&ObjDesc,&MsgDesc,&PutMsgOpts, BufferLength,Buffer,
&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQOD ObjDesc;           /*Object descriptor */
MQMD MsgDesc;           /*Message descriptor */
MQPMO PutMsgOpts;       /*Options that control the action of MQPUT1 */
MQLONG BufferLength;    /*Length of the message in Buffer */
MQBYTE Buffer[n];       /*Message data */
MQLONG CompCode;        /*Completion code */
MQLONG Reason;          /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQPUT1'USING HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,
BUFFERLENGTH,BUFFER,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object descriptor
  01 OBJDESC.
  COPY CMQODV.
**Message descriptor
  01 MSGDESC.
  COPY CMQMDV.
**Options that control the action of MQPUT1
```

```
  01 PUTMSGOPTS.
  COPY CMQPMOV.
**Length of the message in Buffer
  01 BUFFERLENGTH PIC S9(9)BINARY.
**Message data
  01 BUFFER PIC X(n).
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PIC S9(9)BINARY.
```

PL/I invocation

```
CALL MQPUT1 (HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,BUFFER,
COMPCODE,REASON);
```

Declare the parameters:

```
dcl Hconn        fixed bin(31);  /*Connection handle */
dcl ObjDesc      like MQOD;      /*Object descriptor */
dcl MsgDesc      like MQMD;      /*Message descriptor */
dcl PutMsgOpts   like MQPMO;     /*Options that control the action of
                                   MQPUT1 */
dcl BufferLength fixed bin(31);  /*Length of the message in Buffer */
dcl Buffer       char(n);        /*Message data */
dcl CompCode     fixed bin(31);  /*Completion code */
dcl Reason       fixed bin(31);  /*Reason code qualifying CompCode */
```

MQSET - Set object attributes

The MQSET call is used to change the attributes of an object represented by a handle. The object must be a queue.

Note: Once you have issued this call, if you issue a rollback, changes made to the WebSphere MQ configuration will be reversed. However, the queue manager's internal control blocks will retain the changes created by the MQSET call. To remove the changes, reissue the MQSET call with the original values.

Syntax

```
MQSET (Hconn,Hobj,SelectorCount,Selectors,IntAttrCount,
IntAttrs,CharAttrLength,CharAttrs,CompCode,Reason)
```

Parameters

The MQSET call has the following parameters:

Hconn (MQHCONN) - input
Connection handle.

The value of Hconn was returned by a previous MQCONN call.

Hobj (MQHOBJ) - input
Object handle.

This handle represents the queue object whose attributes are to be set. The handle was returned by a previous MQOPEN call that specified the MQOO_SET option.

SelectorCount (MQLONG) - input
Count of selectors.

This is the count of selectors that are supplied in the Selectors array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

Selectors (MQLONG×SelectorCount) - input

Array of attribute selectors.

This is an array of SelectorCount attribute selectors; each selector identifies an attribute (integer or character) whose value is to be set.

Each selector must be valid for the type of queue that Hobj represents. Only certain MQIA_* and MQCA_* values are allowed; these values are listed below.

Selectors for all types of queue

MQCA_Q_DESC
Queue description (MQ_Q_DESC_LENGTH).

MQIA_INHIBIT_PUT
Whether put operations are allowed. Can be one of the following values:
MQQA_PUT_ALLOWED
MQQA_PUT_INHIBITED

Selectors for local queues

MQCA_AUTO_REORG_CATALOG
The contents of this field are now treated as comments and may not reflect the actual VSAM catalog containing the reorganization file. For the reorganization process, the VSAM catalog where the reorganization file is defined is now extracted from the system and so no longer needs to be specified in the queue definition.

MQCA_AUTO_REORG_START_TIME
Indicates the time of day, in HHMM format, for the automatic VSAM reorganization to occur following a system restart (MQ_AUTO_REORG_TIME_LENGTH).

MQCA_TRIGGER_CHANNEL_NAME
Channel name for MCA trigger process (MQ_CHANNEL_NAME_LENGTH).

MQCA_TRIGGER_DATA
Trigger user data (MQ_PROCESS_USER_DATA_LENGTH).

MQCA_TRIGGER_PROGRAM_NAME
Program name for trigger process (MQ_TRIGGER_PROGRAM_NAME_LENGTH).

MQCA_TRIGGER_TERM_ID
Terminal identifier for trigger process (MQ_TRIGGER_TERM_ID_LENGTH).

MQCA_TRIGGER_TRANS_ID
Transaction identifier for trigger process (MQ_TRIGGER_TRANS_ID_LENGTH).

MQIA_ACCOUNTING_Q
Queue accounting setting. Can be one of the following values:
MQMON_Q_MGR
MQMON_OFF
MQMON_ON

MQIA_AUTO_REORG_INTERVAL
Indicates the frequency, in minutes, for the automatic VSAM reorganization to occur, after its initial activation.

MQIA_AUTO_REORGANIZATION
Indicates whether the VSAM file hosting a queue should be scheduled for automatic VSAM reorganization. Can be one of the following values:

MQREORG_ENABLED
MQREORG_DISABLED

MQIA_INHIBIT_GET
Whether get operations are allowed. Can be one of the following values:
MQQA_GET_ALLOWED
MQQA_GET_INHIBITED

MQIA_MAX_GLOBAL_LOCKS
Buffer size for queue manager to manage concurrent queue access.

MQIA_MAX_LOCAL_LOCKS
Buffer size for applications to manage concurrent queue access.

MQIA_MAX_MSG_LENGTH
Maximum message length for queue messages.

MQIA_MAX_Q_DEPTH
Maximum allowable queue depth for queues.

MQIA_MAX_Q_TRIGGERS
Maximum number of concurrent trigger instances for a particular queue.

MQIA_MONITORING_Q
Queue monitoring setting. Can be one of the following values:
- MQMON_OFF
- MQMON_Q_MGR
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

MQIA_Q_DEPTH_HIGH_EVENT
Control attribute for queue depth high events.

MQIA_Q_DEPTH_HIGH_LIMIT
High limit for queue depth.

MQIA_Q_DEPTH_LOW_EVENT
Control attribute for queue depth low events.

MQIA_Q_DEPTH_LOW_LIMIT
Low limit for queue depth.

MQIA_Q_DEPTH_MAX_EVENT
Control attribute for queue depth max events.

MQIA_Q_SERVICE_INTERVAL
Limit for queue service interval.

MQIA_Q_SERVICE_INTERVAL_EVENT
Control attribute for queue service interval events.

MQIA_Q_USERS
Maximum number of concurrent opens per queue.

MQIA_SHAREABILITY
Queue shareability mode. Can be one of the following values:
MQQA_SHAREABLE
MQQA_NOT_SHAREABLE

MQIA_STATISTICS_Q
Queue statistics setting. Can be one of the following values:
- MQMON_Q_MGR
- MQMON_OFF
- MQMON_ON

MQIA_TRIGGER_CONTROL
Whether a trigger is required for the queue. Can be one of the following values:
MQTC_OFF
MQTC_ON

MQIA_TRIGGER_RESTART
Indicator for the reactivation of a trigger process. Can be one of the following values:
MQTRIGGER_RESTART_YES
MQTRIGGER_RESTART_NO

MQIA_TRIGGER_TYPE
Trigger event type. Can be one of the following values:
MQTT_NONE
MQTT_FIRST
MQTT_EVERY

MQIA_USAGE
Queue usage. Can be one of the following values:
MQUS_NORMAL
MQUS_TRANSMISSION

Selectors for remote queues

MQCA_REMOTE_Q_MGR_NAME
Name of remote queue manager (MQ_Q_MGR_NAME_LENGTH).

MQCA_REMOTE_Q_NAME
Name of remote queue as known on remote queue manager (MQ_Q_NAME_LENGTH).

MQCA_XMIT_Q_NAME
Name of local transmission queue (MQ_Q_NAME_LENGTH).

Selectors for alias queues

MQCA_BASE_Q_NAME
Name of queue that alias resolves to (MQ_Q_NAME_LENGTH).

MQIA_INHIBIT_GET
Whether get operations are allowed. Can be one of the following values:
MQQA_GET_ALLOWED
MQQA_GET_INHIBITED

IntAttrCount (MQLONG) - input
Count of integer attributes.

This is the number of elements in the IntAttrs array, and must be at least the number of MQIA_* selectors in the Selectors parameter. Zero is a valid value if there are none.

IntAttrs (MQLONG×IntAttrCount) - input
Array of integer attributes.

This is an array of IntAttrCount integer attribute values. These attribute values must be in the same order as the MQIA_* selectors in the Selectors array.

If the IntAttrCount or SelectorCount parameter is zero, IntAttrs is not referred to; in this case, the parameter address passed by programs written in C may be null.

CharAttrLength (MQLONG) - input
Length of character attributes buffer.

This is the length in bytes of the CharAttrs parameter and for WebSphere MQ for z/VSE must zero.

CharAttrs (MQCHAR×CharAttrLength) - input
Character attributes.

This is not referred to by WebSphere MQ for z/VSE. If programs are written in C then the parameter address passed by programs written in C may be null.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode.

If CompCode is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_FAILED:
MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.
MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.
MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.
MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.
MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.
MQRC_INHIBIT_VALUE_ERROR
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.
MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.
MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.
MQRC_NOT_OPEN_FOR_SET
(2040, X'7F8') Queue not open for set.
MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.
MQRC_Q_DELETED
2052, X'804') The queue has been deleted.
MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.
MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.
MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too big.
MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.
MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQSET (Hconn,Hobj,SelectorCount,Selectors,IntAttrCount,IntAttrs,
CharAttrLength,CharAttrs,&CompCode,&Reason);
```

Declare the parameters:

```
MQHCONN Hconn;          /*Connection handle */
MQHOBJ  Hobj;           /*Object handle */
MQLONG  SelectorCount;  /*Count of selectors */
MQLONG  Selectors[n];   /*Array of attribute selectors */
MQLONG  IntAttrCount;   /*Count of integer attributes */
MQLONG  IntAttrs[n];    /*Array of integer attributes */
MQLONG  CharAttrLength; /*Length of character attributes buffer */
MQCHAR  CharAttrs[n];   /*Character attributes */
MQLONG  CompCode;       /*Completion code */
MQLONG  Reason;         /*Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSET'USING HCONN,HOBJ,SELECTORCOUNT, SELECTORS-
TABLE,INTATTRCOUNT,INTATTRS-TABLE,
CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON.
```

Declare the parameters:

```
**Connection handle
  01 HCONN PIC S9(9)BINARY.
**Object handle
  01 HOBJ PIC S9(9)BINARY.
**Count of selectors
  01 SELECTORCOUNT PIC S9(9)BINARY.
**Array of attribute selectors
  01 SELECTORS-TABLE.
     02 SELECTORS PIC S9(9)BINARY OCCURS n TIMES.
**Count of integer attributes
  01 INTATTRCOUNT PIC S9(9)BINARY.
**Array of integer attributes
  01 INTATTRS-TABLE.
     02 INTATTRS PIC S9(9)BINARY OCCURS n TIMES.
**Length of character attributes buffer
  01 CHARATTRLENGTH PIC S9(9)BINARY.
**Character attributes
  01 CHARATTRS PIC X(n).
**Completion code
  01 COMPCODE PIC S9(9)BINARY.
**Reason code qualifying CompCode
  01 REASON PICS9(9)BINARY.
```

PL/I invocation

```
CALL MQSET (HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,
INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON);
```

Declare the parameters:

```
DCL HCONN          FIXED BIN(31);  /*Connection handle */
DCL HOBJ           FIXED BIN(31);  /*Object handle */
DCL SELECTORCOUNT  FIXED BIN(31);  /*Count of selectors */
DCL SELECTORS(N)   FIXED BIN(31);  /*Array of attribute selectors */
DCL INTATTRCOUNT   FIXED BIN(31);  /*Count of integer attributes */
DCL INTATTRS(N)    FIXED BIN(31);  /*Array of integer attributes */
DCL CHARATTRLENGTH FIXED BIN(31);  /*Length of character attributes
                                             buffer */
DCL CHARATTRS      CHAR(N);        /*Character attributes */
DCL COMPCODE       FIXED BIN(31);  /*Completion code */
DCL REASON         FIXED BIN(31);  /*Reason code qualifying CompCode */
```

MQSETMP - Set message property

The MQSETMP call sets or modifies a property of a message handle.

Syntax

```
MQSETMP ( Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength,
Value, CompCode, Reason )
```

Parameters

The MQSETMP call has the following parameters:

Hconn (MQHCONN) - input
This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the Hmsg parameter.

Hmsg (MQHMSG) - input
This is the message handle to be modified. The value was returned by a previous MQCRTMH call.

SetPropOpts (MQSMPO) - input
Control how message properties are set.

This structure allows applications to specify options that control how message properties are set. The structure is an input parameter on the MQSETMP call. See MQSMPO for further information.

Name (MQCHARV) - input
This is the name of the property to set.

PropDesc (MQPD) - input/output
This structure is used to define the attributes of a property, including:
- What happens if the property is not supported.
- What message context the property belongs to.
- What messages the property is copied into as it flows.

See MQPD for further information about this structure.

Type (MQLONG) - input
The data type of the property being set. It can be one of the following:

MQTYPE_BOOLEAN
A boolean. ValueLength must be 4.

MQTYPE_BYTE_STRING
A byte string. ValueLength must be zero or greater.

MQTYPE_INT8
An 8-bit signed integer. ValueLength must be 1.

MQTYPE_INT16
A 16-bit signed integer. ValueLength must be 2.

MQTYPE_INT32
A 32-bit signed integer. ValueLength must be 4.

MQTYPE_FLOAT32
A 32-bit floating-point number. ValueLength must be 4.

MQTYPE_FLOAT64
A 64-bit floating-point number. ValueLength must be 8.

MQTYPE_STRING
A character string. ValueLength must be zero or greater, or the special value MQVL_NULL_TERMINATED.

MQTYPE_NULL
The property exists but has a null value.ValueLength must be zero.

ValueLength (MQLONG) - input
The length in bytes of the property value in the Value parameter. Zero is valid only for null values or for strings or byte strings. Zero indicates that the property exists but that the value contains no characters or bytes.

The value must be greater than or equal to zero or the following special value if the Type parameter has MQTYPE_STRING set:

MQVL_NULL_TERMINATED
The value is delimited by the first null encountered in the string. The null is not included as part of the string. This value is invalid if MQTYPE_STRING is not also set.

Note: The null character used to terminate a string if MQVL_NULL_TERMINATED is set is a null from the character set of the Value.

Value (MQBYTE x ValueLength) - input
The value of the property to be set. The buffer must be aligned on a boundary appropriate to the nature of the data in the value.

In the C programming language, the parameter is declared as a pointer-to-void. The address of any type of data can be specified as the parameter.

If ValueLength is zero, Value is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode (MQLONG) - output
The completion code; it has one of these values:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
The reason code qualifying CompCode.

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If CompCode is MQCC_WARNING:

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid property data type.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_SMPO_ERROR
(2463, X'099F') Set message property options structure not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQSETMP ( Hconn, Hmsg, &SetPropOpts, &Name, &PropDesc, Type,
ValueLength, &Value, &CompCode, &Reason );
```

Declare the parameters:

```
MQHCONN Hconn; /* Connection handle */
MQHMSG Hmsg; /* Message handle */
MQSMPO SetPropOpts; /* Options that control the action of MQSETMP */
MQCHARV Name; /* Property name */
MQPD PropDesc; /* Property descriptor */
MQLONG Type; /* Property data type */
MQLONG ValueLength; /* Length of property value in Value */
MQBYTE Value[n]; /* Property value */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSETMP'USING HCONN,HOBJ,OPTIONS,COMPCODE,REASON.
```

Declare the parameters:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message handle
01 HMSG PIC S9(18) BINARY.
** Options that control the action of MQSETMP
01 SETMSGOPTS.
   COPY CMQSMPOV.
** Property name
01 NAME
   COPY CMQCHRVV.
** Property descriptor
01 PROPDESC.
   COPY CMQPDV.
** Property data type
01 TYPE PIC S9(9) BINARY.
** Length of property value in VALUE
01 VALUELENGTH PIC S9(9) BINARY.
** Property value
01 VALUE PIC X(n).
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSETMP ( Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type,
ValueLength,Value, CompCode, Reason );
```

Declare the parameters:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hmsg char(8); /* Message handle */
dcl SetPropOpts like MQSMPO;
                 /* Options that control the action of MQSETMP */
dcl Name like MQCHARV; /* Property name */
dcl PropDesc like MQPD; /* Property descriptor */
dcl Type fixed bin(31); /* Property data type */
dcl ValueLength fixed bin(31); /* Length of property value in Value */
dcl Value char(n); /* Property value */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

MQSUB - Register subscription

Use the MQSUB call to register the applications subscription to a particular topic.

Syntax

```
MQSUB (Hconn, SubDesc, Hobj, Hsub, Compcode, Reason)
```

Parameters

The MQSUB call has the following parameters:

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN or MQCONNX call.

SubDesc

Type: MQSD - input/output

This is a structure that identifies the object in use that is being registered by the application. See “MQSD – Subscription descriptor” on page 852 for more information.

Hobj Type: MQHOBJ - input/output

This handle represents the access that has been established to obtain the messages sent to this subscription. These messages can either be stored on a specific queue or the queue manager can manage their storage without using a specific queue.

To use a specific queue, you must associate it with the subscription when the subscription is created. You can do this in two ways:

- By using the DEFINE SUB MQSC command and provided that command with the name of a queue object.
- By providing this handle when calling MQSUB with the MQSO_CREATE. If this handle is provided as an input parameter on the call, it must be a valid object handle returned from a previous MQOPEN call of a queue using at least one of the following options:

```
MQOO_INPUT_*
MQOO_BROWSE
MQOO_OUTPUT (if the queue is a remote queue)
```

If this is not the case, the call fails with MQRC_HOBJ_ERROR. It cannot be an object handle to an alias queue that resolves to a topic object. If so, the call fails with MQRC_HOBJ_ERROR.

If the queue manager is to manage the storage of messages sent to this subscription, this should be set when you create the subscription, by using the MQSO_MANAGED option. The queue manager then returns this handle as an output parameter on the call. The handle that is returned is known as a managed handle. If MQHO_NONE is specified but MQSO_MANAGED is not specified, the call fails with MQRC_HOBJ_ERROR.

When a managed handle is returned to you by the queue manager, you can use it on an MQGET call with or without browse options, on an MQINQ call, or on MQCLOSE. You cannot use it on MQPUT, MQSUB, MQSET; attempting to do so fails with MQRC_NOT_OPEN_FOR_OUTPUT, MQRC_HOBJ_ERROR, or MQRC_NOT_OPEN_FOR_SET.

If this subscription is being resumed using the MQSO_RESUME option in the MQSD structure, the handle can be returned to the application in this parameter by setting MQSO_MANAGED to MQHO_NONE. You can do this whether the subscription is using a managed handle or not and it can be useful to provide subscriptions created using DEFINE SUB with the handle to the subscription queue defined on that command. In the case where an administratively created subscription is being resumed, the queue opens with MQOO_INPUT and MQOO_BROWSE.

If you need to specify other options, the application must open the subscription queue explicitly and provide the object handle on the call. If there is a problem opening the queue the call fails with MQRC_INVALID_DESTINATION.

If the Hobj is provided, it must be equivalent to the Hobj in the original MQSUB call. This means if an object handle returned from an MQOPEN call is being provided, the handle must be to the same queue as previously used. If it is not the same queue, the call fails with MQRC_HOBJ_ERROR.

The table summarizes the use of this parameter with various subscription options:

Table 72. The use of Hobj with different subscription options

Options	Hobj	Description
MQSO_CREATE + MQSO_MANAGED	Ignored on input	Creates a subscription with storage of messages managed by the queue manager.
MQSO_CREATE	A valid object handle	Creates a subscription providing a specific queue as the destination for messages.
MQSO_RESUME	MQHO_NONE	Resumes a previously created subscription whether it was managed or not, and has the queue manager return the object handle for use by the application.
MQSO_RESUME	A valid, matching, object handle	Resumes a previously created subscription that uses a specific queue as the destination for messages and use an object handle with specific open options.

Note: MQSO_ALTER is not currently supported in WebSphere MQ for z/VSE.

Whether it was provided or returned, Hobj must be specified on subsequent MQGET call that want to receive the publication messages sent to this subscription.

The Hobj handle is no longer valid when the MQCLOSE call is issued on it, or the application program finishes.

An MQCLOSE of the Hobj handle does not affect the Hsub handle.

Hsub Type: MQHOBJ - output

This handle represents the subscription that has been made. It can be used for two further operations:

- It can be used on a subsequent MQSUBRQ call to request that publications be sent when the MQSO_PUBLICATIONS_ON_REQUEST option has been used when making the subscription.
- It can be used on a subsequent MQCLOSE call to remove the subscription that has been made. The Hsub handle ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. An MQCLOSE of the Hsub handle does not affect the Hobj handle.

This handle cannot be passed to an MQGET call. You must use the Hobj parameter. You cannot use this handle on any WebSphere MQ call other than MQCLOSE or MQSUB. Passing this handle to any other WebSphere MQ call results in MQRC_HOBJ_ERROR.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:
MQCC_OK
Successful completion
MQCC_WARNING
Warning (partial completion)
MQCC_FAILED
Call failed

Reason
Type: MQLONG - output

Language invocations

This call is supported in the following programming languages:

C invocation

MQSUB (Hconn, &SubDesc, &Hobj, &Hsub, &Code, &Reason)

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQSD    SubDesc;  /* Subscription descriptor */
MQHOBJ  Hobj;     /* Object handle */
MQHOBJ  Hsub;     /* Subscription handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUB' USING HCONN, SUBDESC, HOBJ, HSUB, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Subscription descriptor
 01  SUBDESC.
     COPY CMQSDV.
**   Object handle
 01  HOBJ     PIC S9(9) BINARY.
**   Subscription handle
 01  HSUB     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUB (Hconn, SubDesc, Hobj, Hsub, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn    fixed bin(31); /* Connection handle */
dcl SubDesc  like MQSD;     /* Subscription descriptor */
dcl Hobj     fixed bin(31); /* Object handle */
dcl Hsub     fixed bin(31); /* Subscription handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUB,(HCONN,SUBDESC,HOBJ,HSUB,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN    DS      F  Connection handle
SUBDESC  CMQSDA  ,  Subscription descriptor
HOBJ     DS      F  Object handle
HSUB     DS      F  Subscription handle
COMPCODE DS      F  Completion code
REASON   DS      F  Reason code qualifying COMPCODE
```

MQSUBRQ - Subscription request

Use the MQSUBRQ call to make a request for the retained publication, when the subscriber has been registered with MQSO_PUBLICATIONS_ON_REQUEST.

Syntax

```
MQSUBRQ (Hconn, Hsub, Action, SubRqOpts, Compcode, Reason)
```

Parameters

The MQSUBRQ call has the following parameters:

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN or MQCONNX call.

Hsub Type: MQHOBJ - input

This handle represents the subscription for which an update is to be requested. The value of Hsub was returned from a previous MQSUB call.

Action
Type: MQLONG - input

This parameter controls the particular action that is being requested on the subscription. The following value must be specified:

MQSR_ACTION_PUBLICATION

This action requests that an update publication is sent for the specified topic. It can be used only if the subscriber specified the option MQSO_PUBLICATIONS_ON_REQUEST on the MQSUB call when it made the subscription. If the queue manager has a retained publication for the topic, this is sent to the subscriber. If not, the call fails. If an application is sent a publication which was retained, this is indicated by the MQIsRetained message property of that publication.

Since the topic in the existing subscription represented by the Hsub parameter can contain wildcards, the subscriber might receive multiple retained publications.

SubRqOpts
Type: MQSRO - input/output

These options control the action of MQSUBRQ, see "MQSRO - Subscription request options" on page 862 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address instead of specifying the address of an MQSRO structure.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:
MQCC_OK
Successful completion
MQCC_WARNING
Warning (partial completion)
MQCC_FAILED
Call failed

Reason
Type: MQLONG - output

The reason code qualifying CompCode.

Language invocations

This call is supported in the following programming languages:

C invocation

```
MQSUB (Hconn, Hsub, Action, &SubRqOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;   /* Connection handle */
MQHOBJ Hsub;     /* Subscription handle */
MQLONG Action;   /* Action requested by MQSUBRQ */
MQSRO SubRqOpts; /* Options that control the action of MQSUBRQ */
MQLONG CompCode; /* Completion code */
MQLONG Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUBRQ' USING HCONN, HSUB, ACTION, SUBRQOPTS, COMPCODE,
REASON.
```

Declare the parameters as follows:

```
**   Connection handle
01   HCONN    PIC S9(9) BINARY.
**   Subscription handle
01   HSUB     PIC S9(9) BINARY.
**   Action requested by MQSUBRQ
01   ACTION   PIC S9(9) BINARY.
**   Options that control the action of MQSUBRQ
01   SUBRQOPTS.
     COPY CMQSROV.
**   Completion code
01   COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
01   REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUBRQ (Hconn, Hsub, Action, SubRqOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn    fixed bin(31); /* Connection handle */
dcl Hsub     fixed bin(31); /* Subscription handle */
dcl Action   fixed bin(31); /* Action requested by MQSUBRQ */
dcl SubRqOpts like MQSRO;   /* Options that control the action of MQSUBRQ */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUBRQ,(HCONN, HSUB, ACTION, SUBRQOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS      F  Connection handle
HSUB      DS      F  Subscription handle
ACTION    DS      F  Action requested by MQSUBRQ
SUBRQOPTS CMQSROA ,  Options that control the action of MQSUBRQ
COMPCODE  DS      F  Completion code
REASON    DS      F  Reason code qualifying COMPCODE
```

Attributes of WebSphere MQ objects

In WebSphere MQ for z/VSE, the attributes of all objects are as described in the *WebSphere MQ Application Programming Reference* manual, with the following exception:

- Attributes of process definitions do not apply.
- Attributes of AuthInfo definitions do not apply.

The platform constant MQPL_VSE applies, value 27L.

Reason codes

In WebSphere MQ for z/VSE, the reason codes (MQRC_) are described in older versions of the WebSphere MQ Application Programming Reference manual or in Websphere MQ Messages.

Appendix C. Application Programming Guidance

This appendix describes:
- Application environment overview.
- Sample source code overview.
- Application design guidelines.
- Dynamic queues.

Application environment overview

WebSphere MQ application programs need specific objects before they can run successfully. For example, Figure 87 shows an application that removes messages from a queue, processes them, and then sends some results to another queue on the same queue manager.

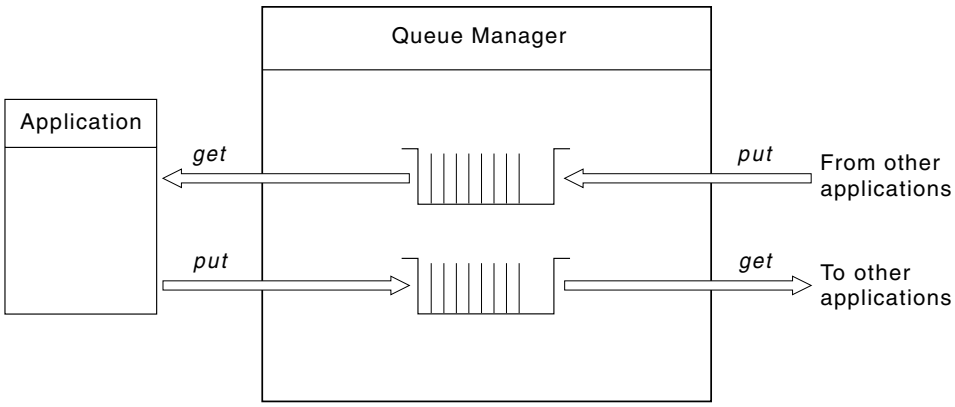


Figure 87. Queues, messages, and applications

Whereas applications can put (using MQPUT) messages on local or remote queues, they can only get (using MQGET) messages directly from local queues.

Before this application can run, these conditions must be satisfied:
- The queue manager must exist and be running.
- The first application queue, from which the messages are to be removed, must be defined.
- The second queue, on which the application puts the messages, must also be defined.
- The application must be able to connect to the queue manager. To do this the application must be linked to the WebSphere MQ for z/VSE MQI object files provided in the installation sublibrary.
- The applications that put the messages on the first queue must also connect to a queue manager. If they are remote, they must also be set up with transmission queues and channels. This part of the system is not shown in Figure 87.

Sample source code overview

One COBOL-language sample trigger program, MQPECHO is provided with WebSphere MQ for z/VSE. The source code for this program is provided in the WebSphere MQ installation sublibrary. Within the source code for MQPECHO, you can find examples that illustrate the use of the MQI calls in a trigger program.

In addition there are three sample programs, TTPTST1, TTPTST2, and TTPTST3. COBOL language copybook files are provided with the distribution file in the installation sublibrary. These files provide examples of all of the MQI calls.

Compiling your application program

The MQI calls are provided in the library PRD2.WMQZVSE.

Compilation

Ensure that you include the PRD2.WMQZVSE library as part of the application phase step.

Developing applications in the C and PL/I programming languages

For CICS, COBOL is the language in which the WebSphere MQ interface is written. Applications written in COBOL for z/VSE have been thoroughly tested with WebSphere MQ. Sample programs and copybooks are provided in COBOL for z/VSE.

However, for a variety of reasons, you may need to write in another programming language. In these cases, you must meet the requirements of the COBOL language interface.

There are no sample programs provided in any other language, however, there are equivalents to the COBOL copybooks to enable applications to be built in other languages.

For the PL/I programming language, the following include files are provided:

CMQEPP.P
: Declares the MQI calls and structures

CMQP.P
: Declares the MQI constants

CMQCFP.P
: Constants for MQI and PCF

CMQXP.P
: Constants for MQI and PCF

For the C programming language, the following header files are provided:

CMQC.H
: MQI header file.

CMQCFC.H
: Constants for MQI and PCF.

CMQXC.H
: Constants for MQI and PCF.

Application design guidelines

One of the key benefits provided by WebSphere MQ is the ability for a distributed application to be developed that is totally independent of the underlying network. This network independence means that there is no need for an application to be aware of:

- The lower levels of the communication protocols, or
- The physical location of other applications on the network.

In order to take full advantage of this network independence, you must choose the queue names used by the application with care.

In particular, you are recommended to use a single logical name only, in your application programs, to refer to each WebSphere MQ queue. For the WebSphere MQ calls, this means only the Queue_Name field is used to identify queues. The use of the queue's fully qualified name (which includes both the Queue_Name field and the Queue_Manager_Name field) is not recommended.

The same is true when addressing WebSphere MQ queues. As the Queue_Manager_Name is typically associated with a particular system, its use implies knowledge of the physical network.

Note: You are strongly recommended to use the Queue_Name field as the only logical queue name. This usage maximizes application flexibility and network independence. The mapping of the queue name in this form to the proper network destination then becomes a configuration issue to be handled by the WebSphere MQ system administrator.

Application syncpoint

Syncpoints allow an application to perform a series of changes, where the changes are treated as though they were a single change.

Syncpoint considerations

Most applications need to access resources of one form or another, and a common requirement is to be able to make a coordinated set of changes to two or more resources.

"Coordinated" means that either all of the changes made to the resources take effect, or none of the changes takes effect. For some applications, queues need to be coordinated. Applications need to be able to get and put messages (and possibly update other resources, for example, databases), and know that either all of the operations take effect, or that none of the operations takes effect.

This set of coordinated operations is called a unit of work. An example of a unit of work is a debit and credit for a funds transfer in a financial application. Both operations must complete, or neither operation must complete, for a valid financial transaction to be completed.

Units of work: A unit of work starts when the first recoverable resource is affected. For message queuing, a unit of works starts when a message get or put occurs under syncpoint control.

The unit of work ends when either the application ends, or when the application declares a syncpoint.

If the unit of work is ended by an application ending, another unit of work can start. One instance of an application can be involved with several sequential units of work.

When a syncpoint is declared, any party (applications and the queue manager) that has interest in the unit of work can vote “yes” to commit the work, or “no”, to back out of the unit of work.

Applications declare syncpoints, and register their votes, by issuing an environment-dependent call. It is advisable that an application should process CICS SYNCPOINT, followed by an MQCMIT call, prior to invoking an MQCLOSE call.

Participation of the MQGET, MQPUT, and MQPUT1 calls in the current unit of work is determined by the environment.

Distributed units of work (involving more than one queue manager) are not supported. A unit of work can contain queuing operations at only one instance of the queue manager.

If a message is put to a remote queue (that is, one on another queuing system), the action of the put request can be within the unit of work on the system that puts the message but the arrival of the message on the target (remote) queue is outside its scope.

The get request for the message on the remote queue can be within the scope of work on that system, but the two units of work are not related by the queue manager.

Putting messages within a unit of work: If an MQPUT or MQPUT1 call participates in the current unit of work, the message is not available for retrieval from the target queue, between the completion of the MQPUT call and the successful completion of the unit of work. The only exception to this rule is if the target queue is within the same unit of work as the one within which it was put.

Only when, and if, the unit of work is committed successfully does the message become generally available.

Any errors detected by the queue manager when the message is put are returned to the application immediately, by means of the completion code and reason code parameters. Errors that can be detected in this way include:
- Message too large for queue.
- Queue full.
- Put requests inhibited for queue.

Failure to put the message does not affect the status of the unit of work, because that message is not part of the unit of work. The application can still commit or backout of the unit of work as required.

However, should an application fail after a message was put successfully within a unit of work, the transaction is backed out.

Getting messages within a unit of work: If an MQGET call participates in the current unit of work, between the completion of the MQGET call and the successful completion of the unit of work, the message remains on the queue but becomes invisible.

Neither the application that retrieved the message, nor any other application serving the queue, can see or obtain the message again. If the unit of work is committed successfully, the message is deleted from the queue. However, if the unit of work is backed out, the message is reinstated in the queue in its original position, and becomes available to the same or another application to retrieve.

Syncpoint and persistence: Persistent messages do not get deleted if the queue manager is restarted. Therefore, they are fully recovered when the queue manager is restarted. Syncpointing by the application causes these records to be in a logical unit of work. Any records that were syncpointed are still recovered if the queue manager is shutdown and restarted.

Non-persistent messages, that is messages placed on a temporary dynamic queue, are removed by the queue manager when WebSphere MQ is stopped and restarted. Non-persistent messages placed on a predefined or permanent dynamic queue are not removed by the queue manager at system restart.

Application rollback

If your application wants to undo what has been done since the beginning of the current logical unit of work, it has to issue the following commands:

```
EXEC CICS SYNCPOINT ROLLBACK
CALL 'MQBACK' USING...
```

This can have the following results:

- Monitoring shows an incorrect queue depth value until the application that rolled back work issues a subsequent MQI call. For this reason, it is recommended that the MQBACK MQI call is used immediately following an application rollback.
- The queue depth and the last queue sequence number (QSN) are not the same. If a message has been rolled back, its queue sequence number is not used again. This is because other applications may have also put messages into the same queue. For example:

```
Transaction A   writes queue sequence number 5
Transaction B                ................. 6
Transaction A                ................. 7
Transaction C                ................. 8
```

  At this point the queue depth is 8. Assume Transaction A rolls back, in which case messages 5 and 7 will be never retrieved. Note that this is not an error. The queue depth is now 6, and the next QSN will be 9.

  From an application point of view this has no impact at all, but can be surprising when using the MQMT dialogs.

  Note: To be able to use SYNCPOINT ROLLBACK, you MUST use a CICS System LOG file, that is, define a CICS JCT. Unpredictable results may occur if the CICS system does not have a CICS system journal defined. Tasks abending or applications issuing CICS ROLLBACK can lead to duplicate messages.

Triggering

Some applications run continuously, and are always available to read a message when it arrives on the application's input queue. However, keeping the application active consumes system resources, even when the application is waiting for a message to arrive. This additional load on the system is not desirable. Instead of

the application running continuously, the application is designed to run only when there are messages to be processed. The queue manager's triggering facility is used to help make this happen.

Overview of triggering

A local queue definition can have a trigger event associated with it when it is defined. This event is defined to activate the MQ trigger API Handler, that is, the MQ02 CICS Transaction.

The trigger API handler does either a CICS LINK to the application program or a CICS START to the application transaction. This is based on whether you defined a program name or a transaction name in the queue definition.

When a trigger type of (E)very is used to trigger a transaction, and the number of messages exceeds the maximum starts specified for the queue, then maintaining the number of active trigger transactions is managed by the WebSphere MQ for z/VSE system monitor (transaction MQSM) at a frequency determined by the system wait interval. This can lead to spikes of trigger tasks starting after every system wait interval. If this poses a problem, it is recommended that triggers be started by specifying an application program name rather than a transaction ID. When this is done, the queue manager will maintain the requested number of maximum starts as each trigger application finishes.

When an application program is entered, an information area is available. This area can be mapped by using the structure defined in the member CMQTMV.C:

1. If the trigger facility specified a program name, this area is passed using the COMMAREA.

   To return to the API handler, you should issue an EXEC CICS RETURN.
2. If the trigger facility specified a transaction name, this information area can be accessed by issuing an EXEC CICS RETRIEVE command.

   Before exiting from the program, you must issue an MQCLOSE command.

Note: In order to perform this function, this transaction ID must be unique in respect to any WebSphere MQ system local queue. Essentially, the WebSphere MQ system queue manager recognizes this transaction ID as a local queue being opened. When this queue is closed fully, this trigger event will be closed, allowing another trigger for this queue to be activated.

Trigger conditions

The queue manager activates a trigger event based on the event type defined for the current queue, against which the MQPUT operation has been requested.

Note: If a non-empty queue is stopped and restarted, the trigger condition suffices, regardless of the trigger event type.

The trigger API handler waits until this MQPUT request has been completed. This implies that the MQPUT request can be successful or unsuccessful, that is, rolled back. The activated trigger application program should perform an MQGET call.

If the result of this MQGET call is an empty condition, that is, MQRC_NO_MSG_AVAILABLE, the original application current logical unit-of-work has been rolled back. It is up to the application trigger program to determine whether to continue to wait or just end.

A trigger event type of "FIRST" generates a trigger event after the queue goes from an empty status to a non-empty one. Therefore, any application triggered in this manner must process the queue until the queue is empty.

A trigger event type of "EVERY" generates a trigger event after every MQPUT call has been completed, up to the maximum number of trigger events specified on the Extended Local Queue Configuration screen. See "Local queue extended definition screen" on page 100 for further information.

Defining a sender channel component

A sender channel component causes the channel to start if there are messages on the transmission queue to be sent to the remote node.

In contrast, a server channel component will not start unless started by a remote requester component, or by manual intervention, even when there are messages to be sent.

On the transmission queue for the sender channel, code the fields as follows:
- Usage Mode - T.
- Trigger Enable - Y.
- Trigger Type - E.
- Max Trigger Starts - 1.
- Transaction ID - <blanks>.
- Program ID - MQPSEND.
- Remote CID - <the name of the channel>.

Note: WebSphere MQ for z/VSE does not support requester channels.

Defining a program to be triggered

This technique is used when an application program is to receive messages from the WebSphere MQ system queue manager in the manner described in "Overview of triggering" on page 946 for a CICS LINK.
- Usage Mode - N.
- Trigger Enable - Y.
- Trigger Type - E or F.
- Max Trigger Starts - 1.
- Transaction ID - <blanks>.
- Program ID - <application program name>.
- Remote CID - <blanks>.
- User data - <optional data for trigger program>.

Defining a transaction to be triggered

"Overview of triggering" on page 946, for CICS START, provides details of how to trigger a program based on its transaction ID. Note, that the transaction should not be invoked outside the trigger mechanism. However, by defining a different transaction name with the same program name, the program can be invoked outside the trigger environment.

Code as follows in the queue definition:
- Usage Mode - N.
- Trigger Enable - Y.
- Trigger Type - E or F.
- Max Trigger Starts - 1.
- Transaction ID - <user Transaction>.
- Program ID - <blanks>.
- Remote CID - <blanks>.

- User data - <optional data for trigger program>.

Queue depth

The QDepth, as displayed in MQMT option 3.1, shows the current number of unread messages on a queue, including uncommitted messages being put to the queue. It does not show uncommitted messages currently being retrieved from the queue. It is the value of the Last Written (LW) pointer less the Last Read (LR) pointer, less any messages within that range which have been retrieved and committed or which were put and rolled back.

The QDepth is maintained by the queue manager using internal tables. These tables are associated with a queue and with each queue object handle open for that queue. Whenever the queue manager is called (by means of an MQI call), it checks to see if a CICS SYNCPOINT or SYNCPOINT ROLLBACK was performed by the application. If either was performed, the queue manager updates its internal tables.

Failure of the application to commit or rollback the unit of work before closing a queue object handle will result in these internal tables not being updated correctly, and lead to the Last Read (LR) and QDepth not being correct.

The Last Read (LR) not being updated can result in additional VSAM I/O as the queue always begins searching for next committed message from the Last Read (LR) pointer. An invalid QDepth may result in triggering of a task when there are no messages for the trigger application to retrieve.

If an application has to perform significant processing following committing a unit of work, but before calling the queue manager again, then an MQCMIT call can be performed to enter the queue manager and update its internal tables. The same is true if the application issues a CICS SYNCPOINT ROLLBACK followed by an MQBACK. Note that in an online application these calls, MQCMIT and MQBACK, do not issue the CICS SYNCPOINT. The SYNCPOINT should be issued explicitly by the application.

Distribution lists

Distribution lists allow you to put a message to multiple destinations in a single MQPUT or MQPUT1 call. Multiple queues can be opened using a single MQOPEN and a message can then be put to each of those queues using a single MQPUT. Some generic information from the MQI structures used for this process can be superseded by specific information relating to the individual destinations included in the distribution list.

When an MQOPEN call is issued, generic information is taken from the Object Descriptor (MQOD). If you specify MQOD_VERSION_2 in the Version field and a value greater than zero in the RecsPresent field, the Hobj can be defined as a handle of a list (of one or more queues) rather than of a queue. In this case, specific information is given through the object records (MQORs), which give details of destination (that is, ObjectName and ObjectQMgrName).

When a message is put on the queues (MQPUT), generic information is taken from the Put Message Option structure (MQPMQ) and the Message Descriptor (MQMD). Specific information is given in the form of Put Message Records (MQQPMRs).

Response Records (MQRR) can receive a completion code and reason code specific to each destination queue.

Opening distribution lists

Use the MQOPEN call to open a distribution list, and use the options of the call to specify what you want to do with the list.

As input to MQOPEN, you must supply:

- A connection handle.
- Generic information in the Object Descriptor structure (MQOD).
- The name of each queue you want to open, using the Object Record structure (MQOR).

The output from the MQOPEN is:

- An object handle that represents your access to the distribution list.
- A generic completion code.
- A generic reason code.
- Response Records (optional), containing a completion code and reason for each destination.

The following notes apply to the use of distribution lists.

1. Fields in the MQOD structure must be set as follows when opening a distribution list:
   - Version must be MQOD_VERSION_2 or greater.
   - ObjectType must be MQOT_Q.
   - ObjectName must be blank or the null string.
   - ObjectQMgrName must be blank or the null string.
   - RecsPresent must be greater than zero.
   - One of the ObjectRecOffset and ObjectRecPtr must be zero and the other nonzero.
   - There must be RecsPresent object records, addressed by either ObjectRecOffset or ObjectRecPtr. The object records must be set to the names of the destination queues to be opened.
   - No more than one of ResponseRecOffset and ResponseRecPtr can be nonzero.
   - If one of ResponseRecOffset and ResponseRecPtr is nonzero, there must be RecsPresent response records present. These are set by the queue manager if the call completes with reason code MQRC_MULTIPLE_REASONS.

   A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that RecsPresent is zero.
2. Only the following open options are valid in the Options parameter:
   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING
3. The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code MQRC_Q_TYPE_ERROR. However, this does not prevent other queues in the list being opened successfully.
4. The completion code and reason parameters are set as follows:

- If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case. For example, if every open succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE, respectively; if every open fails because none of the queues exists, the parameters are set to MQCC_FAILED and MQ_UNKNOWN_OBJECT_NAME.
- If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
  - The completion code parameter is set to MQCC_WARNING, if at least one open succeeded, and to MQCC_FAILED if all failed.
  - The reason code parameter is set to MQRC_MULTIPLE_REASONS. The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

5. When a distribution list has been opened successfully, the handle Hobj returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution lists, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is MQCO_NONE. The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.
6. Each successfully-opened destination in the distribution list counts as a separate handle when checking whether the application has exceeded the permitted maximum number of handles (see the MaxOpen queue-manager attribute). This is true even when two or more of the destinations in the distribution lists actually resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed MaxOpen, the call fails with reason code MQRC_HANDLW_NOT_AVAILABLE.
7. Each destination that is opened successfully has the value of its OpenOutputCount attribute incremented by one. If two or more of the destinations in the distribution list actually resolve to the same physical queue, that queue has its OpenOutputCount attribute incremented by the number of destinations in the distribution list that resolve to that queue.
8. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
9. It is valid for a distribution list to contain only one destination.

Using the MQOD structure

Use the MQOD structure to identify the queues you want to open. To define a distribution list, you must specify MQOD_VERSION_2 in the Version field, a value greater than zero in the RecsPresent field, and MQOT_Q in the ObjectType field. See the WebSphere MQ Application Programming Reference manual for a description of all the fields of the MQOD structure.

Using the MQOR structure

An MQOR structure must be provided for each destination. The structure contains the destination queue and queue manager names. The ObjectName and ObjectQMgrName fields in the MQOD are not used for distribution list. There must be one or more object records. If the ObjectMQMgrName is left blank, the

local queue manager is used. See the WebSphere MQ Application Programming Reference manual for further information about these fields.

You can specify the destination queues in two ways:

1. By using the offset field ObjectRecOffset.

   In this case, the application should declare its own structure containing an MQOD structure, followed by an array of MQOR records (with as many array elements as are needed), and set ObjectRecOffset to the offset of the first element in the array from the start of the MQOD. Care must be taken to ensure that this offset is correct.

   Use of the built-in facilities provided by the programming language is recommended, if these are available in all of the environments in which the application must run. The following illustrates this technique for the COBOL programming language:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
         COPY CMQODV.
      02 MY-MQOR-TABLE OCCURS 100 TIMES.
         COPY CMQORV.

      MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

   Alternatively, the constant MQOD_CURRENT_LENGTH can be used if the programming language does not support the necessary built-in facilities in all of the environments concerned. The following illustrates this technique:

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
      COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

   However, this will work correctly only if the MQOD structure and the array of MQOR records are contiguous; if the compiler inserts skip bytes between the MQOD and the MQOR array, these must be added to the value stored in ObjectRecOffset.

   Using ObjectRecOffset is recommended for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).

2. By using the pointer field ObjectRecPtr.

   In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set ObjectRecPtr to the address of the array. The following illustrates this technique for the C programming language:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

   Using ObjectRecPtr is recommended for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

   Whichever technique is chosen, one of ObjectRecOffset and ObjectRecPtr must be used; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are nonzero.

Using the MQRR structure

These structures are destination specific as each Response Record contains a CompCode and Reason field for each queue of a distribution list. You must use this structure to enable you to distinguish where any problems lie.

For example, if you receive a reason code of MQRC_MULTIPLE_REASONS, and your distribution list contains five destination queues, you will not know which queues the problems apply to if you do not use this structure. However, if you have a completion code and reason code for each destination, you can locate the errors more easily.

See the WebSphere MQ Application Programming Reference manual for further information about the MQRR structure.

Using the MQOPEN options

The following options can be specified when opening a distribution list in WebSphere MQ for z/VSE:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (optional)

A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes.

Putting messages to a distribution list

To put messages to a distribution list, you can use MQPUT or MQPUT1. As input, you must supply:

- A connection handle.
- An object handle. If a distribution list is opened using MQOPEN, the Hobj allows you only to put to the list.
- A message descriptor structure (MQMD). See the WebSphere MQ Application Programming Reference manual for a description of this structure.
- Control information in the form of a put-message option structure (MQPMO).
- Control information in the form of Put Message Records (MQPMR).
- The length of the data contained within the message (MQLONG).
- The message data itself.

The output is:

- A completion code.
- A reason code.
- Response Records (optional).

Using the MQPMR structure

This structure is optional and gives destination-specific information for some fields that you may want to identify differently from those already identified in the MQMD. For a description of these fields, see the WebSphere MQ Application Programming Reference manual.

The content of each record depends on the information given in the PutMsgRecFields field of the MQPMO. For example:

```
typedef struct
{
 MQBYTE24 MsgId;
```

```
 MQBYTE24 CorrelId;
} PutMsgRec;...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

This implies that MsgId and CorrelId are provided for each destination of a distribution list. The Put Message Records are provided as an array.

Using MQPUT1

If you are using MQPUT1, consider the following:

- The values of the ResponseRecOffset and ResponseRecPtr fields must be null or zero.
- The Response Records, if required, must be addressed from the MQOD.

Closing distribution lists

The following points apply if the object being closed is a distribution list:

- The only valid close option for a distribution list is MQCO_NONE; the call fails with reason code MQRC_OPTIONS_ERROR or MQRC_OPTION_NOT_VALID_FOR_TYPE, if any other options are specified.
- When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list. Only the CompCode and Reason parameters of the call are available for diagnostic purposes. If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The CompCode and Reason parameters of the call are then set to return information describing the failure. Thus, it is possible for the completion code to be MQC_FAILED, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

  If there is a failure on more that one queue, it is not defined which failure is reported in the CompCode and Reason parameters.

Object configuration

The queue manager, and relevant queues, must be configured to allow the use of distribution lists.

Queue Manager

The queue manager has the following attributes which affect the operation of distribution lists:

DistLists (MQLONG)

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. The value is one of the following:

- MQDL_SUPPORTED Distribution lists supported.
- MQDL_NOT_SUPPORTED Distribution lists not supported.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call.

MaxQOpen (MQLONG)

Maximum number of open queue handles.

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue, or for an object that is not a queue, uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle,

and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for MaxQOpen.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

The value is in the range 1 through 999 999 999. The default value is determined by the environment.

To determine the value of this attribute, use the MQIA_MAX_HANDLES selector with the MQINQ call.

Queues

Queues have the following attributes which affect the operation of distribution lists:

DistLists (MQLONG)
Distribution list support.

This indicates whether distribution-list messages can be placed on the queue. The attribute is set by a message channel agent (MCA) to inform the local queue manager whether the queue manager at the other end of the channel supports distribution lists. This latter queue manager, called the "partnering queue manager", is the one which next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The attribute is set by the sending MCA whenever it establishes a connection to the receiving MCA on the partnering queue manager. In this way, the sending MCA can cause the local queue manager to place on the transmission queue, only messages which the partnering queue manager is capable of processing correctly.

This attribute is primarily for use with transmission queues, but the processing described is performed regardless of the usage defined for the queue (see the Usage attribute).

The value is one of the following:

MQDL_SUPPORTED
Distribution lists supported.

This indicates that distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

This indicates that distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages will be processed correctly by the partnering queue manager.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Dynamic queues

When an application program issues an MQOPEN call to open a model queue, the queue manager dynamically creates an instance of a local queue with the same attributes as the model queue. Depending on the value of the DefinitionType field of the model queue, the queue manager creates either a temporary or permanent dynamic queue.

Properties of temporary dynamic queues

Temporary dynamic queues have the following properties:

- They hold nonpersistent messages only.
- They are non-recoverable.
- They are deleted when the queue manager is started.
- They are deleted when the application that issued the MQOPEN call which resulted in the creation of the queue closes the queue or terminates.
- If there are any committed messages on the queue, they will be deleted.
- If the queue happens to be in use at this time (by the creating, or another application), the queue is marked as being logically deleted, and is only physically deleted when closed by the last application using the queue.

  Attempts to access a logically deleted queue (other than to close it) fail with reason code MQRC_Q_DELETED.

MQCO_NONE, MQCO_DELETE and MQCO_DELETE_PURGE are all treated as MQCO_NONE when specified on an MQCLOSE call for the corresponding MQOPEN call that created the queue.

Properties of permanent dynamic queues

Permanent dynamic queues have the following properties:

- They hold persistent or nonpersistent messages.
- They are recoverable in the event of system failures.
- They are deleted when an application (not necessarily the one that issued the MQOPEN call which resulted in the creation of the queue) successfully closes the queue using the MQCO_DELETE, or the MQCO_DELETE_PURGE option.
- A close request with the MQCO_DELETE option fails if there are any messages (committed or uncommitted) still on the queue. A close request with the MQCO_DELETE_PURGE option succeeds, even if there are committed messages on the queue (the messages being deleted as part of the close) or if there are uncommitted MQGET, MQPUT, or MQPUT1 calls outstanding against the queue.
- If the delete request is successful, but the queue happens to be in use (by the creating, or another application), the queue is marked as being logically deleted and is only physically deleted when closed by the last application using the queue.
- If an application closing the queue, was not the application that issued the MQOPEN which created the queue, authorization checks are performed against the user.
- They can be deleted in the same way as a normal queue.

Uses of dynamic queues

You can use dynamic queues for:

- Applications that do not require queues to be retained after the application has terminated.
- Applications that require replies to messages to be processed by another application can dynamically create a reply-to queue by opening a model queue. For example, a client application could:
  - Create a dynamic queue.
  - Supply its name in the ReplyToQ field of the message descriptor structure of the request message.
  - Place the request on a queue being processed by a server.
  - The server could then place the reply message on the reply-to queue. Finally, the client could process the reply, and close the reply-to queue with the delete option.

Recommendations for uses of dynamic queues

You should consider the following points when using dynamic queues:

- In a client-server model, each client should create and use its own dynamic reply-to queue. If a dynamic reply-to queue is shared between more than one client, the deletion of the reply-to queue may be delayed because there is uncommitted activity outstanding against the queue, or because the queue is in use by another client. Additionally, the queue may be marked as being logically deleted, and hence inaccessible for subsequent API requests (other than MQCLOSE).
- If your application environment requires that dynamic queues must be shared between applications, you should ensure that the queue is only closed (with the delete option) when all activity against the queue has been committed. This should be by the last user preferably. This ensures that deletion of the queue is not delayed, and should minimize the period that the queue is inaccessible because it has been marked as being logically deleted.

Creating dynamic queues

To create a dynamic queue, you use a template known as a model queue, together with the MQOPEN call. You create a model queue using the WebSphere MQ commands or the master terminal transactions. The dynamic queue you create takes the attributes of the model queue.

When you call MQOPEN, specify the name of the model queue in the ObjectName field of the MQOD structure.

When the call completes, the ObjectName field is set to the name of the dynamic queue that is created. Also, the ObjectQMgrName field is set to the name of the local queue manager. Subsequent operations using the Hobj returned by the MQOPEN call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls.

There are three ways to specify the name of the dynamic queue you create:

- Give the full name you want in the DynamicQName field of the MQOD structure.
- Specify a prefix (fewer than 33 characters) for the name, and allow the queue manager to generate the rest of the name. This means that the queue manager generates a unique name, but you still have some control (for example, you may

want each user to use a certain p refix, or you may want to give a special security classification to queues with a certain prefix in their name). To use this method, specify an asterisk (*) for the last non-blank character of the DynamicQName field. Do not specify a single asterisk (*) for the dynamic queue name.

- Allow the queue manager to generate the full name. To use this method, specify an asterisk (*) in the first character position of the DynamicQName field. WebSphere MQ for z/VSE generates a queue name prefixed by "AMQ.".

For more information about these methods, see the description of the DynamicQName field in the WebSphere MQ Application Programming Reference manual.

Closing dynamic queues

When using the MQCLOSE MQI call, the Options parameter controls how the object is closed. Only permanent dynamic queues can be closed in more than one way, being either retained or deleted; these are queues whose DefinitionType attribute has the value MQQDT_PERMANENT_DYNAMIC.

One (and only one) of the following must be specified:

MQCO_NONE
No optional close processing required. This must be specified for:

- Objects other than queues.
- Predefined queues.
- Temporary dynamic queues (but only in those cases where Hobj is not the handle returned by the MQOPEN call that created the queue).

In all of the above cases, the object is retained and not deleted. If this option is specified for a temporary dynamic queue, the queue is deleted (if it was created by the MQOPEN call that returned Hobj) and any messages that are on the queue are purged. In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

MQCO_DELETE
Delete the queue.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, and there are no messages on the queue.
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj. In this case, all the messages on the queue are purged.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

MQCO_DELETE_PURGE
Delete the queue, purging any messages on it.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue.
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

The following points apply if the object being closed is a dynamic queue (either permanent or temporary):

- For a dynamic queue, the options MQCO_DELETE or MQCO_DELETE_PURGE can be specified regardless of the options specified on the corresponding MQOPEN call.
- When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. After a dynamic queue has been deleted, any call (other than MQCLOSE) that attempts to reference the queue using a previously acquired Hobj handle fails with reason code MQRC_Q_DELETED.

  Be aware that although a deleted queue cannot be accessed by applications, the queue is not removed from the system, and associated resources are not freed, until such time as all handles that reference the queue have been closed.
- When a dynamic queue is deleted, if the Hobj handle specified on the MQCLOSE call is not the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier is authorized to delete the queue. This check is not performed if:
  - The handle specified is the one returned by the MQOPEN call that created the queue.
  - The queue being deleted is a temporary dynamic queue.
- When a temporary dynamic queue is closed, if the Hobj handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated. If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but this does not cause the units of work to fail. However, as described above, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

Queue definition types

This indicates how the queue was defined. The value is one of the following:

MQQDT_PREDEFINED
Predefined permanent queue.

The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created, for example, using the DEFINE MQSC command, and can be deleted using the DELETE MQSC command. Predefined queues cannot be created from model queues.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the DefinitionType attribute.

This type of queue can be deleted using the MQCLOSE call.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the DefinitionType attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

Dynamic queue name

DynamicQName is a field of the MQOD data structure passed to the MQOPEN call.

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when ObjectName specifies the name of a model queue; in all other cases DynamicQName is ignored.

The characters that are valid in the name are the same as those for ObjectName (see above), except that an asterisk is also valid (see below). A name that is completely blank (or one in which only blanks appear before the first null character) is not valid if ObjectName is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to appear in the first character position, in which case the name consists solely of the characters generated by the queue manager.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment. On WebSphere MQ for z/VSE, the value is 'AMQ.*'.

The value is a blank-padded string.

Message grouping and segmentation

This section describes message grouping and segmentation features as applicable to WebSphere MQ for z/VSE.

Key concepts and definitions

The following definitions describe key concepts of message grouping and segmentation.

Message group
A group of logical messages. Logical grouping of messages allows applications to group messages that are similar and to ensure the sequence of the messages.

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in

the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there will be more than n physical messages in the group.

Message segment
One of a number of segments of a message that is too large either for the application or for the queue manager to handle.

Physical message
This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (MsgId field in MQMD), although this is not enforced by the queue manager.

Logical message
This is a single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message. But where logical messages are extremely large, system constraints may make it advisable or necessary to split a logical message into two or more physical messages, called segments.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (GroupId field in MQMD), and the same message sequence number (MsgSeqNumber field in MQMD). The segments are distinguished by differing values for the segment offset (Offset field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message groups

Messages can occur within groups. This allows ordering of messages, and segmentation of large messages within the same group.

The hierarchy within a group is as follows: group, logical message, segment.

Group This is the highest level in the hierarchy and is identified by a GroupId. It consists of one or more messages that contain the same GroupId. These messages can be stored anywhere on the queue.

Note: The term "message" is used here to denote one item on a queue, such as would be returned by a single MQGET that does not specify MQGMO_COMPLETE_MSG.

Logical message
Logical messages within a group are identified by the GroupId and MsgSeqNumber fields. The MsgSeqNumber starts at 1 for the first message within a group, and if a message is not in a group, the value of the field is 1. Logical messages within a group can be used to:

- Ensure ordering (if this is not guaranteed under the circumstances in which the message is transmitted).
- Allow applications to group together similar messages (for example, those that must all be processed by the same server instance).

Each message within a group consists of one physical message, unless it is split into segments. Each message is logically a separate message, and only the GroupId and MsgSeqNumber fields in the MQMD need bear any relationship to other messages in the group. Other fields in the MQMD are independent; some may be identical for all messages in the group whereas others may be different. For example, messages in a group may have different format names, CCSIDs, encodings, and so on.

Segment

Segments are used to handle messages that are too large for either the putting or getting application or the queue manager including intervening queue managers through which the message passes).

A segment of a message is identified by the GroupId, MsgSeqNumber, and Offset fields. The Offset field starts at zero for the first segment within a message.

Each segment consists of one physical message that may or may not belong to a group. A segment is logically part of a single message, so only the MsgId, Offset, and SegmentFlag fields in the MQMD should differ between separate segments of the same message.

Message segmentation

For putting and destructively getting, it is assumed that the MQPUT or MQGET calls always operate within a unit of work. It is strongly recommended that this technique is always used, to reduce the possibility of incomplete groups being present in the network. Single-phase commit by the queue manager is assumed, but of course other coordination techniques are equally valid.

Also, in the getting applications, it is assumed that if multiple servers are processing the same queue, each server executes similar code, so that one server never fails to find a message or segment that it expects to be there (because it had specified MQGMO_ALL_MSGS_AVAILABLE or MQGMO_ALL_SEGMENTS_AVAILABLE earlier).

Segmentation and reassembly by queue manager

This is the simplest scenario, in which one application puts a message to be retrieved by another. The message may be large: not too large for either the putting or the getting application to handle in a single buffer, but possibly too large for the queue manager or a queue on which the message is to be put.

The only changes necessary for these applications are for the putting application to authorize the queue manager to perform segmentation if necessary,

```
PMO.Options = (existing options)
MQPUT MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
```

and for the getting application to ask the queue manager to reassemble the message if it has been segmented:

```
GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
MQGET
```

The application buffer must be large enough to contain the reassembled message (unless the MQGMO_ACCEPT_TRUNCATED_MSG option is included).

If data conversion is necessary, it may have to be done by the getting application specifying MQGMO_CONVERT. This should be straightforward because the data conversion exit is presented with the complete message. Attempting to do data conversion in a sender channel will not be successful if the message is segmented, and the format of the data is such that the data-conversion exit cannot carry out the conversion on incomplete data.

Application segmentation

This example shows how to segment a single large message.

Application segmentation is used for two main reasons:

- Queue-manager segmentation alone is not adequate because the message is too large to be handled in a single buffer by the applications.
- Data conversion must be performed by sender channels, and the format is such that the putting application needs to stipulate where the segment boundaries are to be in order for conversion of an individual segment to be possible.

However, if data conversion is not an issue, or if the getting application always uses MQGMO_COMPLETE_MSG, queue-manager segmentation can also be allowed by specifying MQMF_SEGMENTATION_ALLOWED. In our example, the application segments the message into four segments:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

If you do not use MQPMO_LOGICAL_ORDER, the application must set the Offset and the length of each segment. In this case, logical state is not maintained automatically.

The getting application cannot, or chooses not to, guarantee to have a buffer that will hold any reassembled message. It must therefore be prepared to process segments individually.

For messages that are segmented, this application does not want to start processing one segment until all of the segments that constitute the logical message are present. MQGMO_ALL_SEGMENTS_AVAILABLE is therefore specified for the first segment. If you specify MQGMO_LOGICAL_ORDER and there is a current logical message, MQGMO_ALL_SEGMENTS_AVAILABLE is ignored.

Once the first segment of a logical message has been retrieved, MQGMO_LOGICAL_ORDER is used to ensure that the remaining segments of the logical message are retrieved in order.

No consideration is given to messages within different groups. If such messages do occur, they are processed in the order in which the first segment of each message appears on the queue.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
```

```
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Application segmentation of logical messages

The messages must be maintained in logical order in a group, and some or all of them may be so large that they require application segmentation.

In our example, a group of four logical messages is to be put. All but the third message are large, and require segmentation which is performed by the putting application:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP       | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP       | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP       | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP       | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP       | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

In the getting application, MQGMO_ALL_MSGS_AVAILABLE is specified on the first MQGET. This means that no messages or segments of a group are retrieved until the entire group is available. When the first physical message of a group has been retrieved, MQGMO_LOGICAL_ORDER is used to ensure that the segments and messages of the group are retrieved in order:

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the
      GroupStatus and SegmentStatus information to see what has
      been returned */
   ...

MQCMIT
```

Note: If you specify MQGMO_LOGICAL_ORDER and there is a current group, MQGMO_ALL_MSGS_AVAILABLE is ignored.

Logical and physical ordering

Messages on queues can occur in physical or logical order:

Physical
This is the order in which messages arrive on a queue.

Logical
This is when all of the messages and segments within a group are in their logical sequence, adjacent to each other, in the position determined by the physical position of the first item belonging to the group.

These physical and logical orders may differ because:

- Groups can arrive at a destination at similar times from different applications, therefore losing any distinct physical order.
- Even within a single group, messages may get out of order due to rerouting or delay of some of the messages in the group.

For example, these messages may appear in the following logical order on a queue:
Message A (not in a group).
Logical message 1 of group Y.
Logical message 2 of group Y.
Segment 1 of (last) logical message 3 of group Y.
(Last) segment 2 of (last) logical message 3 of group Y.
Logical message 1 of group Z.
(Last) logical message 2 of group Z.
Message B (not in a group).

The physical order, however, may be entirely different. The physical position of the first item within each group determines the logical position of the whole group. For example, if groups Y and Z arrived at similar times, and message 2 of group Z overtook message 1 of the same group, the physical order would look like:
Message A (not in a group).
Logical message 1 of group Y.
Logical message 2 of group Z.
Logical message 2 of group Y.
Segment 1 of (last) logical message 3 of group Y.
(Last) segment 2 of (last) logical message 3 of group Y.
Logical message 1 of group Z.
Message B (not in a group).

When getting messages, you can specify MQGMO_LOGICAL_ORDER to retrieve messages in logical rather than physical order.

If you issue an MQGET call with MQGMO_BROWSE_FIRST and MQGMO_LOGICAL_ORDER, subsequent MQGET calls with MQGMO_BROWSE_NEXT must also specify this option. Conversely, if the MQGET with MQGMO_BROWSE_FIRST does not specify MQGMO_LOGICAL_ORDER, neither must the following MQGETs with MQGMO_BROWSE_NEXT.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that the queue manager retains for MQGET calls that remove messages from the queue. When MQGMO_BROWSE_FIRST is specified, the queue manager ignores the group and segment information for browsing, and scans the queue as though there were no current group and no current logical message.

Note: Special care is needed if an MQGET call is used to browse beyond the end of a message group (or logical message not in a group) when MQGMO_LOGICAL_ORDER is not specified. For example, if the last message in the group happens to precede the first message in the group on the queue, using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with MsgSeqNumber set to 1 (to find the first message of the next group) would return again the first message in the group already browsed. This could happen immediately, or a number of MQGET calls later (if there are intervening groups).

The possibility of an infinite loop can be avoided by opening the queue twice for browse:

- Use the first handle to browse only the first message in each group.
- Use the second handle to browse only the messages within a specific group.
- Use the MQMO_* options to move the second browse cursor to the position of the first browse cursor, before browsing the messages in the group.
- Do not use the MQGMO_BROWSE_NEXT browse beyond the end of a group.

For further information about this, see the *WebSphere MQ Application Programming Reference*.

For most applications you will probably choose either logical or physical ordering when browsing. However, if you want to switch between these modes, remember that when you first issue a browse with MQGMO_LOGICAL_ORDER, your position within the logical sequence is established.

If the first item within the group is not present at this time, the group you are in is not considered to be part of the logical sequence.

Once the browse cursor is within a group, it can continue within the same group, even if the first message is removed. Initially though, you can never move into a group using MQGMO_LOGICAL_ORDER where the first item is not present.

Grouping logical messages

There are two main reasons for using logical messages in a group:

- The messages may need to be processed in the correct order.
- Each of the messages in a group may need to be processed in a related way.

In either case, retrieval of the entire group must be carried out by the same getting application instance.

For example, assume that the group consists of four logical messages. The putting application looks like this:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

The getting application chooses not to start processing any group until all of the messages within it have arrived. MQGMO_ALL_MSGS_AVAILABLE is therefore specified for the first message in the group; the option is ignored for subsequent messages within the group.

Once the first logical message of the group is retrieved, MQGMO_LOGICAL_ORDER is used to ensure that the remaining logical messages of the group are retrieved in order.

So, the getting application looks like this:

```
/* Wait for the first message in a group, or a message not in
   a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
```

```
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

Putting and getting a group that spans units of work

In the previous case, messages or segments cannot start to leave the node (if its destination is remote) or start to be retrieved until all of the group has been put and the unit of work is committed. This may not be what you want if it takes a long time to put the whole group, or if queue space is limited on the node. To overcome this, the group can be put in several units of work.

If the group is put within multiple units of work, it is possible for some of the group to commit even when a failure of the putting application occurs. The application must therefore save status information, committed with each unit of work, which it can use after a restart to resume an incomplete group. The simplest place to record this information is in a STATUS queue. If a complete group has been successfully put, the STATUS queue is empty.

If segmentation is involved, the logic is similar. In this case, the StatusInfo must include the Offset.

Here is an example of putting the group in several units of work:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

If all the units of work have been committed, the entire group has been put successfully, and the STATUS queue is empty. If not, the group must be resumed at the point indicated by the status information. MQPMO_LOGICAL_ORDER cannot be used for the first put, but can thereafter.

Restart processing looks like this:

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...
```

```
else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

From the getting application, you may want to start processing the messages in a group before the whole group has arrived. This improves response times on the messages within the group, and also means that storage is not required for the entire group.

For recovery reasons, each message must be retrieved within a unit of work. However, in order to realize the above benefits, several units of work must be used for each group of messages.

As with the corresponding putting application, this requires status information to be recorded somewhere automatically as each unit of work is committed. Again, the simplest place to record this information is on a STATUS queue. If a complete group has been successfully processed, the STATUS queue is empty.

Note: For intermediate units of work, you can avoid the MQGET calls from the STATUS queue by specifying that each MQPUT to the status queue is a segment of a message (that is, by setting the MQMF_SEGMENT flag), instead of putting a complete new message for each unit of work. In the last unit of work, a final segment is put to the status queue specifying MQMF_LAST_SEGMENT, and then the status information is cleared with an MQGET specifying MQGMO_COMPLETE_MSG. During restart processing, instead of using a single MQGET to get a possible status message, browse the status queue with MQGMO_LOGICAL_ORDER until you reach the last segment (that is, until no further segments are returned). In the first unit of work after restart, also specify the offset explicitly when putting the status segment. In the following example, we consider only messages within a group. It is assumed that the application's buffer is always large enough to hold the entire message, whether or not the message has been segmented. MQGMO_COMPLETE_MSG is therefore specified on each MQGET. The same principles apply if segmentation is involved (in this case, the StatusInfo must include the Offset).

For simplicity, we assume that a maximum of 4 messages should be retrieved within a single UOW:

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
```

```
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options =
           MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

If all of the units of work have been committed, then the entire group has been retrieved successfully, and the STATUS queue is empty. If not, then the group must be resumed at the point indicated by the status information. MQGMO_LOGICAL_ORDER cannot be used for the first retrieve, but can thereafter.

Restart processing looks like this:

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group id with those retrieved from
      the status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID |
                            MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT |
                    MQGMO_WAIT | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...

      /* Have retrieved last message or 4 messages  */
      /* Update status message if not last in group */
      MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
      if ( GroupStatus == MQGS_MSG_IN_GROUP )
         StatusInfo = GroupId,MsgSeqNumber from MQMD
```

```
   MQPUT (StatusInfo to STATUS queue) PMO.Options =
        MQPMO_SYNCPOINT
MQCMIT
msgs = 0
```

Reports and segmented messages

If a message is segmented and you ask for reports to be generated, you may receive more reports than you would have done had the message not been segmented.

WebSphere MQ-generated reports

If you segment your messages or allow the queue manager to do so, there is only one case in which you can expect to receive a single report for the entire message. This is when you have requested only COD reports, and you have specified MQGMO_COMPLETE_MSG on the getting application.

In other cases your application must be prepared to deal with several reports; usually one for each segment.

Note: If you segment your messages, and you need only the first 100 bytes of the original message data to be returned, you must change the setting of the report options to ask for reports with no data for segments that have an offset of 100 or more. If you do not do this, and you leave the setting so that each segment requests 100 bytes of data, and you retrieve the report messages with a single MQGET specifying MQGMO_COMPLETE_MSG, the reports assemble into a large message containing 100 bytes of read data at each appropriate offset. If this happens, you need a large buffer or you need to specify MQGMO_ACCEPT_TRUNCATED_MSG.

Application-generated reports

If your application generates reports, you should always copy the WebSphere MQ headers that are present at the start of the original message data to the report message data. Then add none, 100 bytes, or all of the original message data (or whatever other amount you would normally include) to the report message data.

You can recognize the WebSphere MQ headers that must be copied by looking at the successive Format names, starting with the MQMD nd continuing through any headers present. The following Format names indicate these WebSphere MQ headers:

MQMDE
MQDLH
MQXQH
MQIIH
MQH*

MQH* means any name starting with the characters MQH.

The Format name occurs at specific positions for MQDLH and MQXQH, but for the other WebSphere MQ headers it occurs at the same position. The length of the header is contained in a field that also occurs at the same position for MQMDE, MQIMS and all MQH* headers.

If you are using a Version 1 of the MQMD, and you are reporting on a segment, or a message in a group, or a message for which segmentation is allowed, the report

data must start with an MQMDE. You should set the OriginalLength field to the length of the original message data excluding the lengths of any WebSphere MQ headers that you find.

Retrieval of reports

If you ask for COA or COD reports, you can ask for them to be reassembled for you with MQGMO_COMPLETE_MSG. An MQGET with MQGMO_COMPLETE_MSG is satisfied when enough report messages (of a single type, for example COA, and with the same GroupId) are present on the queue to represent one complete original message. This is true even if the report messages themselves do not contain the complete original data; the OriginalLength field in each report message gives the length of original data represented by that report message, even if the data itself is not present.

This technique can be used even if there are several different report types present on the queue (for example, both COA and COD), because an MQGET with MQGMO_COMPLETE_MSG reassembles report messages only if they have the same Feedback code. Note, however, that you cannot normally use the technique for exception reports, since in general these have different Feedback codes.

You can use this technique to get a positive indication that the entire message has arrived. However, in most circumstances you need to cater for the possibility that some segments arrive while others may generate an exception (or expiry, if you have allowed this). You cannot use MQGMO_COMPLETE_MSG in this case because in general you may get different Feedback codes for different segments and, as noted above, you may get more than one report for a given segment. You can, however, use MQGMO_ALL_SEGMENTS_AVAILABLE.

To allow for this you may need to retrieve reports as they arrive, and build up a picture in your application of what happened to the original message. You can use the GroupId field in the report message to correlate reports with the GroupId of the original message, and the Feedback field to identify the type of each report message. The way in which you do this depends on your application requirements.

One approach is as follows:

1. Ask for COD reports and exception reports.
2. After a specific time, check whether a complete set of COD reports has been received using MQGMO_COMPLETE_MSG. If so, your application knows that the entire message has been processed.

   If not, and exception reports relating to this message are present, the problem should be handled just as for unsegmented messages, though provision must also be made for 'orphan' segments to be cleaned up at some point.

   If there are segments for which there are no reports of any kind, the original segments (or the reports) may be waiting for a channel to be reconnected, or the network may be overloaded at some point. If no exception reports at all have been received (or if you think that the ones you have may be temporary only), you may decide to let your application wait a little longer.

As before, this is similar to the considerations you have when dealing with unsegmented messages, except that you must also consider the possibility of 'orphan' segments which have to be cleaned up.

If the original message is not critical (for example, if it is a query, or a message that can be repeated later), set an expiry time to ensure that orphan segments are removed.

Back-level queue managers

When a report is generated by a queue manager that supports segmentation, but is received on a queue manager that does not support segmentation, the MQMDE structure (which identifies the Offset and OriginalLength represented by the report) is always included in the report data, in addition to zero, 100 bytes, or all of the original data in the message.

However, if a segment of a message passes through a queue manager that does not support segmentation, you should be aware that if a report is generated there, the MQMDE structure in the original message will be treated purely as data. It will not therefore be included in the report data if zero bytes of the original data have been requested. Without the MQMDE, the report message may not be useful.

You should therefore request at least 100 bytes of data in reports if there is a possibility that the message might travel through a back-level queue manager.

Message properties

Use message properties to allow an application to select messages to process, or to retrieve information about a message without accessing MQMD or MQRFH2 headers. They also facilitate communication between Websphere MQ and JMS applications.

A message property is data associated with a message, consisting of a textual name and a value of a particular type. Message properties are used by message selectors to filter publications to topics or to selectively get messages from queues. Message properties can be used to include business data or state information without having to store it in the application data. Applications do not have to access data in the MQ Message Descriptor (MQMD) or MQRFH2 headers because fields in these data structures can be accessed as message properties using Message Queue Interface (MQI) function calls.

The use of message properties in WebSphere MQ mimics the use of properties in JMS. This means that you can set properties in a JMS application and retrieve them in a procedural WebSphere MQ application, or the other way round. To make a property available to a JMS application, assign it the prefix "usr"; it is then available (without the prefix) as a JMS message user property. For example, the Websphere MQ property usr.myproperty (a character string) is accessible to a JMS application using the JMS call:

```
message.getStringProperty('myproperty')
```

Note that a property with the prefix "usr" can contain only a single U+002E (".") character. A property with no prefix and no U+002E (".") character is treated as if it had the prefix "usr". Conversely, a user property set in a JMS application can be accessed in a WebSphere MQ application by adding the "usr." prefix to the property name inquired on in an MQINQMP call.

Message properties and message length

Use the queue manager attribute MaxPropertiesLength to control the size of the properties that can flow with any message in a WebSphere MQ queue manager.

In general, when you use MQSETMP to set properties, the size of a property is the length of the property name in bytes, plus the length of the property value in bytes as passed into the MQSETMP call. It is possible for the character set of the

property name and the property value to change during transmission of the message to its destination because these can be converted into Unicode. In this case the size of the property may change.

On an MQPUT or MQPUT1 call, properties of the message do not count towards the length of the message for the queue and the queue manager, but they do count towards the length of the properties as perceived by the queue manager (whether they were set using the message property MQI calls or not).

If the size of the properties exceeds the maximum properties length, the message is rejected with MQRC_PROPERTIES_TOO_BIG. Because the size of the properties is dependent on its representation, you should set the maximum properties length at a gross level.

It is possible for an application to successfully put a message with a buffer that is larger than the value of MaxMsgLength, if the buffer includes properties. This is because, even when represented as MQRFH2 elements, message properties do not count towards the length of the message. The MQRFH2 header fields add to the properties length only if one or more folders are contained and every folder in the header contains properties. If one or more folders are contained in the MQRFH2 header and any folder does not contain properties, the MQRFH2 header fields count towards the message length instead.

On an MQGET call, properties of the message do not count towards the length of the message as far as the queue and the queue manager are concerned. However, because the properties are counted separately it is possible that the buffer returned by an MQGET call is larger than the value of the MaxMsgLength attribute.

Do not have your applications query the value of MaxMsgLength and then allocate a buffer of this size before calling MQGET; instead, allocate a buffer you consider large enough. If the MQGET fails, allocate a buffer guided by the size of the DataLength parameter.

The DataLength parameter of the MQGET call now returns the length in bytes of the application data and any properties returned in the buffer you have provided, if a message handle is not specified in the MQGMO structure.

The Buffer parameter of the MQPUT call now contains the application message data to be sent and any properties represented in the message data.

There is a length limit of 4 MB for message properties, excluding the message descriptor or extension for each message.

The size of a property in its internal representation is the length of the name, plus the size of its value, plus some control data for the property. There is also some control data for the set of properties after one property is added to the message.

Property names

A property name is a character string. Certain restrictions apply to its length and the set of characters that can be used.

A property name is a case-sensitive character string, limited to +4095 characters unless otherwise restricted by the context. This limit is contained in the MQ_MAX_PROPERTY_NAME_LENGTH constant.

If you exceed this maximum length when using a message property MQI call, the call fails with reason code MQRC_PROPERTY_NAME_LENGTH_ERR.

As there is no maximum property name length in JMS, it is possible for a JMS application to set a valid JMS property name that is not a valid WebSphere MQ property name when stored in an MQRFH2 structure.

In this case, when parsed, only the first 4095 characters of the property name are used; the following characters are truncated. This could cause an application using selectors to fail to match a selection string, or to match a string when not expecting to, since more than one property may truncate to the same name. When a property name is truncated, WebSphereMQ issues an error log message.

All property names must follow the rules defined by the Java Language Specification for Java Identifiers, with the exception that Unicode character U+002E (".") is permitted as part of the name, but not the start. The rules for Java Identifiers equate to those contained in the JMS specification for property names.

White space characters and comparison operators are prohibited. Embedded nulls are allowed in a property name, but not recommended. If you use embedded nulls, this prevents the use of the MQVS_NULL_TERMINATED constant when used with the MQCHARV structure to specify variable length strings.

Keep property names simple because applications can select messages based on the property names and the conversion between the character set of the name and of the selector may cause the selection to fail unexpectedly.

WebSphere MQ property names use character U+002E (".") for logical grouping of properties. This divides up the namespace for properties. Properties with the prefixes shown here, in any mixture of lowercase or uppercase are reserved for use by the product:

mcd
jms
usr
mq
sib
wmq
Root
Body

A good way to avoid name clashes is to ensure that all applications prefix their message properties with their Internet domain name. For example, if you are developing an application using domain name "ourcompany.com" you could name all properties with the prefix "com.ourcompany". This naming convention also allows for easy selection of properties; for example, an application can inquire on all message properties starting "com.ourcompany.%".

See “Property name restrictions” for further information about the use of property names.

Property name restrictions

When you name a property, you must observe certain rules.

These restrictions apply to property names:

- A property must not begin with these strings:

"JMS" Reserved for use by WebSphere MQ classes for JMS.
"usr.JMS"
Invalid.

The only exceptions to this are the properties shown in Table 73 which provide synonyms for JMS properties:

Table 73. Data types of properties

Property	Synonym for
JMSCorrelationID	Root.MQMD.CorrelId or jms.Cid
JMSDeliveryMode	Root.MQMD.Persistence or jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root.MQMD.Expiry or jms.Exp
JMSMessageID	Root.MQMD.MsgId
JMSPriority	Root.MQMD.Priority or jms.Pri
JMSRedelivered	Root.MQMD.BackoutCount
JMSReplyTo (a string encoded as a URI)	Root.MQMD.ReplyToQ or Root.MQMD.ReplyToQMgr or jms.Rto
JMSTimestamp	Root.MQMD.PutDate or Root.MQMD.PutTime or jms.Tms
JMSType	mcd.Type or mcd.Set or mcd.Fmt
JMSXAppID	Root.MQMD.PutApplName
JMSXDeliveryCount	Root.MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId or jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber or jms.Seq
JMSXUserID	Root.MQMD.UserIdentifier

These synonyms allow an MQI application to access JMS properties in a similar fashion to a WebSphere MQ classes for JMS client application. Of these properties, only JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID, and JMSXGroupSeq can be set using the MQI.

Note that the JMS_IBM_* properties available from within WebSphere MQ classes for JMS are not available using the MQI. The fields that the JMS_IBM_* properties reference can be accessed in other ways by MQI applications.

- A property must not be called, in any mixture of lowercase or uppercase, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" or "ESCAPE". These are the names of SQL keywords used in selection strings.
- A property beginning "mq" (except "mq_usr"), "jms", "mcd", "usr", or "sib" (in any mixture of lowercase or uppercase) can only contain a single "." character (U+002E).
- Two "." characters must contain other characters in between; you cannot have an empty point in the hierarchy. Similarly a property name cannot end in a "." character.
- If an application sets the property "a.b" and then the property "a.b.c", it is unclear whether in the hierarchy "b" contains a value or another logical grouping. Such a hierarchy is "mixed content" and this is not supported. Setting a property that causes mixed content is not allowed.

These restrictions are enforced by the validation mechanism as follows:

- Property names are validated when setting a property using the MQSETMP call, if validation was requested when the message handle was created. If an attempt

to validate a property is undertaken and fails due to an error in the specification of the property name, the completion code is MQCC_FAILED with reason:
  – MQRC_PROPERTY_NAME_ERROR for reasons 1-4
  – MQRC_MIXED_CONTENT_NOT_ALLOWED for reason 5
- The names of properties specified directly as MQRFH2 elements are not guaranteed to be validated by the MQPUT call.

Message descriptor fields as properties

Most message descriptor fields can be treated as properties. The property name is constructed by adding a prefix to the name of the message descriptor field.

If an MQI application wants to identify a message property contained in a message descriptor field (for example, in a selector string or using the message property APIs), use the syntax shown in Table 74.

Table 74. Message descriptor field syntax when identifying a message property

Property name	Message descriptor field
Property name Message descriptor field Root.MQMD.<Field>>	<Field>

Specify <Field> with the same case as for the MQMD structure fields in the C language declaration. For example, the property name Root.MQMD.AccountingToken accesses the AccountingToken field of the message descriptor.

The StrucId and Version fields of the message descriptor are not accessible using the syntax shown in Table 74.

Message descriptor fields are never represented in an MQRFH2 header as for other properties.

If the message data starts with an MQMDE that is honored by the queue manager, the MQMDE fields can be accessed using the Root.MQMD.<Field> notation shown in Table 74. In this case, the MQMDE fields are treated as logically part of the MQMD from a properties perspective. See “MQMDE specified on MQPUT and MQPUT1 calls” on page 806.

Property data types and values

A property can be a boolean, a byte string, a character string, or a floating-point or integer number. The property can store any valid value in the range of the data type unless otherwise restricted by the context.

The data type of a property value must be one of these values:
MQBOOL
MQBYTE[ ]
MQCHAR[ ]
MQFLOAT32
MQFLOAT64
MQINT8
MQINT16
MQINT32

A property can exist but have no defined value; In this case, it is a null property. A null property is different from a byte or character string property (MQBYTE[ ] and MQCHAR[ ] respectively) that has a defined but empty value; that is, one with a zero-length value. Byte string is not a valid property data type in JMS or XMS. It is recommended that you do not use byte string properties in the <usr>> folder.

Appendix D. Sample JCL and programs

This appendix provides sample JCL to run WebSphere MQ for z/VSE utility programs, and sample source code that uses the WebSphere MQ MQI. Program samples are provided in COBOL, C, and PL/I.

Sample JCL

This section provides sample JCL for running the MQPUTIL, MQPEXCIC, and MQPMQSC utility programs.

Sample JCL for MQPUTIL

```
* ** JOB JNM=MQJUTILY,DISP=D,CLASS=A
* ** LST DISP=H,CLASS=Q,PRI=3
// JOB MQJUTILY - Execute WebSphere MQ z/VSE Batch Utility Program.
* ------------------------------------------------------------------*
*      I M P O R T A N T    I M P O R T A N T  I M P O R T A N T    *
*                                                                   *
*   Please change :                                                 *
*                 "* ** JOB" to  "* $$ JOB"                         *
*                 "* ** LST" to  "* $$ LST"                         *
*                 "* ** EOJ" to  "* $$ EOJ"                         *
*                                                                   *
* ------------------------------------------------------------------*
*   This job executes MQPUTIL to access the CONFIGURATION file      *
*                                                                   *
*   This file is a sample and needs modification to suit the        *
*   users environment.                                              *
*                                                                   *
* ------------------------------------------------------------------*
* Licensed Materials - Property of IBM                              *
*                                                                   *
* 5655-U97                                                          *
* Copyright IBM Corp. 2008                                          *
*                                                                   *
* US Government Users Restricted Rights - Use, duplication or       *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
* ------------------------------------------------------------------*
* ------------------------------------------------------------------*
*      SELECT ONE OF THE FOLLOWING SYSIPT CARD OPTIONS              *
*      & INSERT IT AFTER // EXEC MQPUTIL ...                        *
* ------------------------------------------------------------------*
*    col 1.................20.......                                *
*        RESET MSN          00000002                                *
*        PRINT MESSAGES                                             *
*        PRINT CONFIG                                               *
*        PRINT LOG                                                  *
*        PRINT LOG FROMQ system.log                                 *
* ------------------------------------------------------------------*
// LIBDEF PHASE,SEARCH=(PRD2.WMQZVSE,PRD2.SCEEBASE)
// DLBL CONFIG,'WMQZVSE.MQFCNFG',,VSAM,CAT=MQMCAT
// DLBL INLOG,'WMQZVSE.MQFLOG',,VSAM,CAT=MQMCAT
// ASSGN SYS005,SYSLST
/. C
/. C if using PRINT LOG FROMQ then ensure following SETPARM is
/. C set to the batch interface id of the required queue manager
/. C
// SETPARM MQBISRV='mqbiserv'
// EXEC MQPUTIL,SIZE=MQPUTIL
/*RESET MSN         00000002
```

```
/*RESET CHECKPOINT  00000002
/*PRINT CONFIG
/*PRINT LOG
/*PRINT LOG FROMQ system.log
/*
/&
* ** EOJ
```

Sample JCL for MQPEXCIC

```
// JOB MQJEXCIC
// ID USER=userid,PWD=userpwd
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,prd2.sceebase)
// ASSGN SYS005,SYSLST
// EXEC MQPEXCIC,PARM='vtam-applid',OS390
CHANNEL(MY.CHANNEL) DLQSTORE(Y)  ENABLE(Y)
/*
/&
```

Sample JCL for MQPMQSC

```
// JOB MQSCRUN
// SETPARM MQBISRV='mqbiserv'
// LIBDEF *,SEARCH=(PRD2.WMQZVSE,prd2.sceebase)
// EXEC MQPMQSC,SIZE=MQPMQSC
*
*  Define a local queue.
*
DEFINE QLOCAL(EXAMPLE.Q)                      +
       DESCR('Example queue')                 +
       CICSFILE(MQFO001)                      +
       USAGE(NORMAL)                          +
       SHARE                                  +
       PUT(ENABLED)                           +
       GET(ENABLED)                           +
*
       MAXDEPTH(5000)                         +
       MAXMSGL(2000)                          +
       MAXQUSER(100)                          +
       MAXGLOCK(200)                          +
       MAXLLOCK(200)                          +
*
       NOTRIGGER                              +
       MAXTRIGS(1)                            +
       NOTRIGREST                             +
*
       QSVCIEV(NONE)                          +
       QSVCINT(0)                             +
       QDPMAXEV(DISABLED)                     +
       QDPHIEV(DISABLED)                      +
       QDEPTHHI(0)                            +
       QDPLOEV(DISABLED)                      +
       QDEPTHLO(0)                            +
*
       REORG(DISABLED)                        +
       REORGINT(0000)                         +
       REORGTI(0000)
/*
/&
```

Sample programs

WebSphere MQ for z/VSE provides a number of source code samples. These are described in the following table:

Table 75. Sample program files

File	Language	Description
DCHFMT4.Z	C	Message data conversion exit
MQBICALL.Z	COBOL	Batch interface MQI program
MQBISTOP.Z	COBOL	Batch interface stop program
MQPCHNX.Z	COBOL	Generic channel exit
MQPECHO.Z	COBOL	Trigger program example
MQPSAXE.Z	C	WMQ API Exit program
TTMTST3.Z	Assembler	BMS map program for TTPTST3
TTPTST1.Z	COBOL	CICS MQI transaction
TTPTST2.Z	COBOL	CICS MQI program
TTPTST3.Z	COBOL	Driver program for TTPTST2

In addition to these sample programs, the following program listings are provided to illustrate the use of the message queue interface (MQI) from programming languages: COBOL, C and PL/I.

Sample COBOL MQI program

```
 IDENTIFICATION DIVISION.
*****************************************************************
*                                                               *
* Program name: MQICPUTC                                        *
*                                                               *
* Description: Sample COBOL program that puts messages          *
*              to a message queue (example using MQPUT)         *
* <START_COPYRIGHT>                                             *
* Licensed Materials - Property of IBM                          *
*                                                               *
* 5655-U97                                                      *
* 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.                                                     *
* <END_COPYRIGHT>                                               *
*                                                               *
*****************************************************************
*                                                               *
* Function:                                                     *
*                                                               *
*                                                               *
*   MQICPUTC is a sample COBOL program to put messages on       *
*   a message queue, and is an example of the use of MQPUT      *
*                                                               *
*      -- target queue name received from screen, i.e.          *
*                                                               *
*         <tranid> <target-queue>                               *
*                                                               *
*      -- open the target queue, put a single message and       *
*         close the queue                                       *
*                                                               *
*      -- writes a message for each MQI reason other than       *
```

```
*         MQCC-OK; stops if there is an MQI failure           *
*                                                             *
*    Program logic:                                           *
*         RECEIVE target queue name from screen               *
*         MQCONNect to default queue manager                  *
*         MQOPEN target queue for OUTPUT                      *
*         MQPUT a message                                     *
*         MQCLOSE target queue                                *
*         MQDISConnect from queue manager                     *
*                                                             *
*                                                             *
***************************************************************
 PROGRAM-ID. 'MQICPUTC'.

***************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
**  Declare MQI structures needed
* MQI named constants
 01 MY-MQ-CONSTANTS.
    COPY CMQV.
* Object Descriptor
 01 OBJECT-DESCRIPTOR.
    COPY CMQODV.
* Message Descriptor
 01 MESSAGE-DESCRIPTOR.
    COPY CMQMDV.
* Put message options
 01 PMOPTIONS.
    COPY CMQPMOV.
** note, sample uses defaults where it can
 01 QM-NAME                    PIC X(48) VALUE SPACES.
 01 HCONN                      PIC S9(9) BINARY.
 01 Q-HANDLE                   PIC S9(9) BINARY.
 01 OPTIONS                    PIC S9(9) BINARY.
 01 COMPLETION-CODE            PIC S9(9) BINARY.
 01 OPEN-CODE                  PIC S9(9) BINARY.
 01 CON-REASON                 PIC S9(9) BINARY.
 01 REASON                     PIC S9(9) BINARY.
 01 SEND-REASON                PIC 9999.
 01 SEND-TEXT                  PIC X(60).
 01 BUFFER                     PIC X(60).
 01 TRANSID                    PIC X(4).
 01 RECEIVE-BUFFER             PIC X(60).
 01 BUFFER-LENGTH              PIC S9(9) BINARY.
 01 RECEIVE-LENGTH             PIC S9(9) BINARY.
 01 TARGET-QUEUE               PIC X(48).


***************************************************************
 PROCEDURE DIVISION.
 P0.

***************************************************************
*                                                             *
*    Get name of target queue via RECEIVE                     *
*                                                             *
***************************************************************
     EXEC CICS RECEIVE
               INTO(RECEIVE-BUFFER)
               LENGTH(LENGTH OF RECEIVE-BUFFER)
               NOHANDLE
     END-EXEC.

     UNSTRING RECEIVE-BUFFER DELIMITED BY ALL SPACES INTO
              TRANSID
```

```
              TARGET-QUEUE.

*****************************************************************
*                                                               *
*   Connect to default queue manager                            *
*                                                               *
*****************************************************************
     CALL 'MQCONN'
      USING QM-NAME, HCONN,
      COMPLETION-CODE, CON-REASON.

*      report reason and stop if it failed
     IF COMPLETION-CODE NOT = MQCC-OK
       MOVE CON-REASON TO SEND-REASON
       STRING 'MQCONN ended with reason code ' SEND-REASON
         DELIMITED BY LOW-VALUE
         INTO SEND-TEXT
       PERFORM ERRORS
       END-IF.

*****************************************************************
*                                                               *
*   Open the target message queue for output                    *
*                                                               *
*****************************************************************
 OPENS.
     MOVE TARGET-QUEUE TO MQOD-OBJECTNAME.
     MOVE MQOO-OUTPUT TO OPTIONS.
     CALL 'MQOPEN'
      USING HCONN, OBJECT-DESCRIPTOR,
      OPTIONS, Q-HANDLE,
      OPEN-CODE, REASON.

*      report reason, if any; stop if failed
     IF OPEN-CODE NOT = MQCC-OK
       MOVE REASON TO SEND-REASON
       STRING 'MQOPEN ended with reason code ' SEND-REASON
         DELIMITED BY LOW-VALUE
         INTO SEND-TEXT
       PERFORM ERRORS
       END-IF.

*****************************************************************
*                                                               *
*   Put message to the target queue                             *
*                                                               *
*****************************************************************
 PUTS.
     MOVE LENGTH OF BUFFER TO BUFFER-LENGTH.
     MOVE MQFMT-STRING TO MQMD-FORMAT.
     MOVE 'This is a test message' TO BUFFER.
     CALL 'MQPUT'
      USING HCONN, Q-HANDLE,
      MESSAGE-DESCRIPTOR, PMOPTIONS,
      BUFFER-LENGTH, BUFFER,
      COMPLETION-CODE, REASON.

*      report reason, if failed
     IF COMPLETION-CODE NOT = MQCC-OK
       MOVE REASON TO SEND-REASON
       STRING 'MQPUT ended with reason code ' SEND-REASON
         DELIMITED BY LOW-VALUE
         INTO SEND-TEXT
       PERFORM ERRORS
       END-IF.

     EXEC CICS SYNCPOINT END-EXEC.
```

```
*****************************************************************
*                                                               *
*   Close the target queue                                      *
*                                                               *
*****************************************************************
 CLOSES.
     MOVE MQCO-NONE TO OPTIONS.
     CALL 'MQCLOSE'
      USING HCONN, Q-HANDLE, OPTIONS,
      COMPLETION-CODE, REASON.

*      report reason, if any
     IF COMPLETION-CODE NOT = MQCC-OK
       MOVE REASON TO SEND-REASON
       STRING 'MQCLOSE ended with reason code ' SEND-REASON
         DELIMITED BY LOW-VALUE
         INTO SEND-TEXT
       PERFORM ERRORS
       END-IF.

*****************************************************************
*                                                               *
*  Disconnect from queue manager (if not previously connected) *
*                                                               *
*****************************************************************
 DISCS.
     IF CON-REASON NOT = MQRC-ALREADY-CONNECTED
       CALL 'MQDISC'
        USING HCONN, COMPLETION-CODE, REASON

*      report reason, if any
     IF COMPLETION-CODE NOT = MQCC-OK
       MOVE REASON TO SEND-REASON
       STRING 'MQDISC ended with reason code ' SEND-REASON
         DELIMITED BY LOW-VALUE
         INTO SEND-TEXT
       PERFORM ERRORS
       END-IF.

*****************************************************************
*                                                               *
*  Indicate success, and return.                                *
*                                                               *
*****************************************************************
 OVER.
     STRING 'Message put to ' TARGET-QUEUE
       DELIMITED BY LOW-VALUE
       INTO SEND-TEXT
     EXEC CICS SEND
               FROM   (SEND-TEXT)
               LENGTH (LENGTH OF SEND-TEXT)
               ERASE
     END-EXEC.

     EXEC CICS RETURN END-EXEC.

*****************************************************************
*                                                               *
*   Send error text to the screen and terminate.                *
*                                                               *
*****************************************************************
 ERRORS.
     EXEC CICS SEND
               FROM   (SEND-TEXT)
               LENGTH (LENGTH OF SEND-TEXT)
               ERASE
```

```
           END-EXEC.

           EXEC CICS RETURN END-EXEC.

      ****************************************************************
      *                                                              *
      * END OF MQICPUTC                                              *
      *                                                              *
      ****************************************************************
```

Sample C MQI program

```
/****************************************************************************/
/*                                                                          */
/* Program name: MQICINQX                                                   */
/*                                                                          */
/* Description: Sample C program that demonstrates the MQINQ call.          */
/*                                                                          */
/* <START_COPYRIGHT>                                                        */
/* Licensed Materials - Property of IBM                                     */
/*                                                                          */
/* 5655-U97                                                                 */
/* 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.                                                                */
/* <END_COPYRIGHT>                                                          */
/****************************************************************************/
/*                                                                          */
/* Function:                                                                */
/*                                                                          */
/*                                                                          */
/*   MQICINQX is a sample C program that demonstrates how to use            */
/*   the MQINQ call to get information about a queue.                       */
/*                                                                          */
/*   This sample connects to a queue manager, opens a queue,                */
/*   inquires on its attributes, closes the queue and disconnects           */
/*   from the queue manager.                                                */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*   MQICINQX expects input parameters provided via the screen.             */
/*                                                                          */
/*   i.e.                                                                   */
/*                                                                          */
/*       <transid> <source-queue>                                           */
/*                                                                          */
/*                                                                          */
/****************************************************************************/


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

/* Defines */
 #define FALSE             0
 #define TRUE              1
 #define MAX_NO_MSGS       10
 #define MAX_MSG_LEN       80
 #define MAX_INQ_CNT       3
 #define MAX_INT_CNT       2
```

```
 /* Function prototypes */
 void mqStrCpy(char *target, char *source, int len);


/*****************************************************************/
/* This program connects to a queue manager, opens a source queue,  */
/* inquires on its attributes, closes it, and disconnects from the  */
/* queue manager. The source queue name is read from the screen.    */
/*****************************************************************/
 int main(int argc, char **argv)
 {
    /*
     * Local variables.
     */
     int      msgno;
     char     msg[MAX_NO_MSGS][MAX_MSG_LEN];
     char     buf[80];
     short    recvlen;
     short    sendlen;
     MQLONG   selcnt;
     MQLONG   seltab[MAX_INQ_CNT];
     MQLONG   iattrcnt;
     MQLONG   iattrtab[MAX_INT_CNT];
     MQLONG   cattrlen;
     MQCHAR  *cattrtab;
     MQHCONN  Hcon;
     MQHOBJ   Hobj;
     MQLONG   O_options;
     MQLONG   C_options;
     MQLONG   CompCode;
     MQLONG   Reason;
     MQCHAR48 QMName;
     MQOD     od  = { MQOD_DEFAULT };
     struct   tagRBUF
     {
       char   transid[4];
       char   filler;
       char   srcq[MQ_Q_NAME_LENGTH];
     } recvbuf;


    /*
     * Initialize local variables.
     */
     msgno = 0;
     Hcon  = MQHC_UNUSABLE_HCONN;
     Hobj  = MQHO_UNUSABLE_HOBJ;
     memset(&msg, 0, (MAX_NO_MSGS * MAX_MSG_LEN));
     strcpy(&msg[msgno++][0], "Program begins");

    /*
     * Receive source queue name.
     */
     recvlen = sizeof(struct tagRBUF);
     memset(&recvbuf.srcq, 0, MQ_Q_NAME_LENGTH);
     EXEC CICS RECEIVE
               INTO(&recvbuf)
               LENGTH(recvlen)
               NOHANDLE;

    /*
     * Connect to queue manager.
     */
     memset(QMName, ' ', MQ_Q_MGR_NAME_LENGTH);
     MQCONN(QMName,
            &Hcon,
```

```
       &CompCode,
       &Reason);

if ( CompCode != MQCC_OK )
{
    sprintf(msg[msgno++],
            "MQCONN returned (%ld,%ld)",
            CompCode, Reason);
    goto ProgExit;
}

/*
 * Open source queue.
 */
mqStrCpy(od.ObjectName, recvbuf.srcq, MQ_Q_NAME_LENGTH);
O_options = MQOO_INQUIRE;
MQOPEN(Hcon,
       &od,
       O_options,
       &Hobj,
       &CompCode,
       &Reason);

if ( CompCode != MQCC_OK )
{
    sprintf(msg[msgno++],
            "MQOPEN returned (%ld,%ld)\n",
            CompCode, Reason);
    goto ProgExit;
}

/*
 * Inquire on some queue attributes.
 */
selcnt    = MAX_INQ_CNT;
seltab[0] = MQIA_MAX_Q_DEPTH;
seltab[1] = MQIA_MAX_MSG_LENGTH;
seltab[2] = MQCA_Q_DESC;
iattrcnt  = MAX_INT_CNT;
cattrlen  = MQ_Q_DESC_LENGTH;
cattrtab  = (char *)&buf;
MQINQ(Hcon,
      Hobj,
      selcnt,
      seltab,
      iattrcnt,
      iattrtab,
      cattrlen,
      cattrtab,
      &CompCode,
      &Reason);

if ( CompCode != MQCC_OK )
{
    sprintf(msg[msgno++],
            "MQINQ returned (%ld,%ld)\n",
            CompCode, Reason);
    goto ProgExit;
}

/*
 * Report queue attributes.
 */
buf[MQ_Q_DESC_LENGTH] = 0;
sprintf(msg[msgno++],
        "Maximum queue depth (%ld)\n", iattrtab[0]);
sprintf(msg[msgno++],
```

```
                "Maximum message length (%ld)\n", iattrtab[1]);
        sprintf(msg[msgno++], "Description (%s)\n", buf);


    ProgExit:
       /*
        * Close the source queue.
        */
        if ( Hobj != MQHO_UNUSABLE_HOBJ )
        {
            C_options = MQCO_NONE;
            MQCLOSE(Hcon,
                    &Hobj,
                    C_options,
                    &CompCode,
                    &Reason);
            if ( CompCode != MQCC_OK )
            {
                sprintf(msg[msgno++],
                        "MQCLOSE returned (%ld,%ld)\n",
                        CompCode, Reason);
            }
        }

       /*
        * Disconnect from queue manager.
        */
        if ( Hcon != MQHC_UNUSABLE_HCONN )
        {
            MQDISC(&Hcon, &CompCode, &Reason);
            if ( CompCode != MQCC_OK )
            {
                sprintf(msg[msgno++],
                        "MQDISC returned (%ld,%ld)\n",
                        CompCode, Reason);
            }
        }

       /*
        * Terminate.
        */
        strcpy(&msg[msgno++][0], "Program ends");
        sendlen = msgno * 80;
        EXEC CICS SEND
                  FROM(&msg)
                  LENGTH(sendlen)
                  ERASE;

        EXEC CICS RETURN;
    }

    /*********************************************************************/
    /* This function copies a source string to a target string up to     */
    /* the first null, and then pads the rest of the target string       */
    /* with blanks.                                                      */
    /*********************************************************************/
     void mqStrCpy(char *target, char *source, int len)
     {
         int   i, j;

         for ( i=0; i<len; i++ )
         {
             if ( *source == 0x00 )
             {
                 for ( j=0; j<(len-i); j++ )
                     *target++ = ' ';
                 break;
```

```
        }
        else
        {
            *target++ = *source++;
        }
    }

    return;
 }

/*********************************************************************/
/*  End of MQICINQX                                                  */
/*********************************************************************/
```

Sample PL/I MQI program

```
*process langlvl(os),macro,not('^'),or('|');
 /*********************************************************************/
 /*                                                                   */
 /* Program name: MQICGETP                                            */
 /*                                                                   */
 /* Description:  Sample PL/I program for WebSphere MQ that gets      */
 /*               messages from a message queue.                      */
 /* <START_COPYRIGHT>                                                 */
 /* Licensed Materials - Property of IBM                              */
 /*                                                                   */
 /* 5655-U97                                                          */
 /* 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.                                                         */
 /* <END_COPYRIGHT>                                                   */
 /*********************************************************************/
 /*                                                                   */
 /* Function:                                                         */
 /*                                                                   */
 /*   MQICGETP is a sample PL/I program to get messages from a        */
 /*   message queue, and is an example of the use of MQGET.           */
 /*                                                                   */
 /*   -- Receives source queue name from screen.                      */
 /*                                                                   */
 /*   -- Opens the source queue, gets a message, displays it on       */
 /*      the screen, and closes the queue.                            */
 /*                                                                   */
 /*   -- Writes a message for each MQI failure                        */
 /*                                                                   */
 /*                                                                   */
 /* Program logic:                                                    */
 /*                                                                   */
 /*    RECEIVE source queue name from screen                          */
 /*    MQCONN connect to default queue manager                        */
 /*    MQOPEN open queue for input                                    */
 /*    MQGET get next message, remove from queue                      */
 /*    print message                                                  */
 /*    MQCLOSE close queue                                            */
 /*    MQDISC disconnect from queue manager                           */
 /*                                                                   */
 /*********************************************************************/
 /*                                                                   */
 /* MQICGETP has 2 parameters:                                        */
 /*                                                                   */
 /*   - transid                                                       */
 /*   - source queue name                                             */
 /*                                                                   */
 /*   i.e. <transid> <source-queue>                                   */
 /*                                                                   */
```

```
/*********************************************************************/
MQICGET: PROCEDURE OPTIONS(MAIN NOEXECOPS);

/*********************************************************************/
/* Builtin functions                                                 */
/*********************************************************************/
dcl (low
    ,substr
    ,addr
    ,index
    ,null
    ) builtin;

/*********************************************************************/
/* Message Queue Interface (MQI) structures and constants            */
/*********************************************************************/
%include syslib(cmqp);
%include syslib(cmqxp);
%include syslib(cmqcfp);
%include syslib(cmqepp);

/*********************************************************************/
/* Working variables                                                 */
/*********************************************************************/
dcl QName char(MQ_Q_NAME_LENGTH);      /* Queue name                 */
dcl QMName char(MQ_Q_MGR_NAME_LENGTH); /* Queue manager name         */
dcl recvBuf char(60);                  /* Receive buffer             */
dcl charPos fixed bin(15);             /* char position              */
dcl recvLen fixed bin(15);             /* Receive buffer length      */
dcl hCon fixed bin(31);                /* handle to connection       */
dcl hObj fixed bin(31);                /* handle to object           */
dcl options fixed bin(31);             /* options                    */
dcl reason fixed bin(31);              /* reason code                */
dcl connReason fixed bin(31);          /* MQCONN reason code         */
dcl compCode fixed bin(31);            /* completion code            */
dcl openCompCode fixed bin(31);        /* MQOPEN completion code     */
dcl msg char(100) varying;             /* message                    */
dcl msgBuf char(100);                  /* message buffer             */
dcl msgBufLen fixed bin(31);           /* message buffer length      */
dcl msgLen fixed bin(31);              /* message length received    */
dcl outBuf (10) char(80);              /* output buffer              */
dcl outCnt fixed bin(15);              /* output message count       */
dcl outLen fixed bin(15);              /* output buffer length       */

/*********************************************************************/
/* Declare MQI structures needed, using defaults                     */
/*********************************************************************/
dcl 1 od like mqod;                    /* object descriptor          */
dcl 1 md like mqmd;                    /* message descriptor         */
dcl 1 gmo like mqgmo;                  /* get message options        */

/*********************************************************************/
/* Receive the source queue name from the screen.                    */
/*********************************************************************/
outCnt = 1;
outBuf(outCnt) = 'Program begins';
EXEC CICS RECEIVE
          INTO(recvbuf)
          LENGTH(recvlen)
          NOHANDLE;

charPos = index(recvbuf, ' ');
QName = substr(recvbuf, charPos+1, recvlen-charPos);

outCnt = outCnt + 1;
outBuf(outCnt) = 'Source queue is ' || QName;
```

```
/*********************************************************************/
/* Connect to queue manager                                          */
/*********************************************************************/
QMName = '';
call mqconn(QMName, hCon, compCode, connReason);

if compCode = MQCC_FAILED then
do;
   outCnt = outCnt + 1;
   outBuf(outCnt) = 'MQCONN ended with reason code ' || connReason;
   go to ProgExit;
end;

/*********************************************************************/
/* Open the source message queue for input                           */
/*********************************************************************/
options = MQOO_INPUT_SHARED;
od.ObjectName = QName;

call mqopen(hCon, od, options, hObj, openCompCode, reason);

if openCompCode = MQCC_FAILED then
do;
   outCnt = outCnt + 1;
   outBuf(outCnt) = 'MQOPEN ended with reason code ' || reason;
   go to ProgExit;
end;

/*********************************************************************/
/* Set up some things for the MQGET                                  */
/*********************************************************************/
msgBufLen = 100;
gmo.Options = MQGMO_WAIT + MQGMO_CONVERT;
gmo.WaitInterval = 15000;              /* 15 sec limit for waiting   */

/*********************************************************************/
/* Get a message from the queue                                      */
/*********************************************************************/

md.MsgId = MQMI_NONE;
md.CorrelId = MQCI_NONE;

call mqget(hCon, hObj, md, gmo, msgBufLen, msgBuf, msgLen,
           compCode, reason);

if compCode = MQCC_FAILED then
do;
   outCnt = outCnt + 1;
   outBuf(outCnt) = 'MQGET ended with reason code ' || reason;
   go to ProgExit;
end;

/*********************************************************************/
/*   Display the message                                             */
/*********************************************************************/
outCnt = outCnt + 1;
outBuf(outCnt) = 'Message -> ' || msgBuf;

EXEC CICS SYNCPOINT;

ProgExit:

/*********************************************************************/
/* Close queue if opened                                             */
/*********************************************************************/
if openCompCode ^= MQCC_FAILED then
do;
```

```
   options = 0;                         /* no close options           */

   call mqclose(hCon, hObj, options, compCode, reason);

   if compCode = MQCC_FAILED then
   do;
     outCnt = outCnt + 1;
     outBuf(outCnt) = 'MQCLOSE ended with reason code ' || reason;
   end;
end;

/*****************************************************************/
/* Disconnect from queue manager if not already connected        */
/*****************************************************************/
if connReason ^= MQRC_ALREADY_CONNECTED then
do;
   call mqdisc(hCon, compCode, reason);

   if compCode = MQCC_FAILED then
   do;
     outCnt = outCnt + 1;
     outBuf(outCnt) = 'MQDISC ended with reason code ' || reason;
   end;
end;

outCnt = outCnt + 1;
outBuf(outCnt) = 'Program ends';
outLen = outCnt * 80;
EXEC CICS SEND
          FROM(outBuf)
          LENGTH(outLen)
          ERASE;

EXEC CICS RETURN;

END MQICGET;
```

Appendix E. Example configuration - WebSphere MQ for z/VSE Version 3.0.0

This appendix gives an example of how to set up communication links from WebSphere MQ for z/VSE to WebSphere MQ products on the following platforms:

- OS/2
- Windows NT
- AIX
- HP-UX
- AT&T GIS UNIX (This platform has become NCR UNIX SVR4 MP-RAS, R3.0)
- Sun Solaris
- OS/400®
- z/OS without CICS

It describes the parameters needed for an LU 6.2 and TCP/IP connection. Once the connection is established, you need to define some channels to complete the configuration. This is described in "WebSphere MQ for z/VSE configuration" on page 996.

Configuration parameters for an LU 6.2 connection

Table 76 presents a worksheet listing all the parameters needed to set up communication from z/VSE to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter" in conjunction with the worksheet in the *WebSphere MQ Intercommunication* book for the platform to which you are connecting.

Configuration worksheet

Use the following worksheet to record the values you will use for this configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet in the *WebSphere MQ Intercommunication* book. The examples that follow in this chapter refer back to the values in the ID column of this table. The entries in the Parameter Name column are explained in "Explanation of terms" on page 993.

Table 76. Configuration worksheet for z/VSE using APPC

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
1	Network ID		**NETID**	
2	Node name		**VSEPU**	
3	Local LU name		**VSELU**	
4	Local Transaction Program name		**MQ01**	**MQ01**
5	LAN destination address		**400074511092**	
Connection to an OS/2 system The values in this section of the table must match those used in the table for OS/2 in the *WebSphere MQ Intercommunication* book, as indicated.				

Table 76. Configuration worksheet for z/VSE using APPC (continued)

ID	Parameter Name	Reference	Example Used	User Value
6	Connection name		**OS2**	
7	Group name		**EXAMPLE**	
8	Session name		**OS2SESS**	
9	Netname	6	**OS2LU**	
Connection to a Windows NT system The values in this section of the table must match those used in the table for Windows NT in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**WNT**	
7	Group name		**EXAMPLE**	
8	Session name		**WNTSESS**	
9	Netname	5	**WINNTLU**	
Connection to an AIX system The values in this section of the table must match those used in the table for AIX in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**AIX**	
7	Group name		**EXAMPLE**	
8	Session name		**AIXSESS**	
9	Netname	4	**AIXLU**	
Connection to an HP-UX system The values in this section of the table must match those used in the table for HP-UX in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**HPUX**	
7	Group name		**EXAMPLE**	
8	Session name		**HPUXSESS**	
9	Netname	5	**HPUXLU**	
Connection to an AT&T GIS UNIX system The values in this section of the table must match those used in the table for GIS UNIX in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**GIS**	
7	Group name		**EXAMPLE**	
8	Session name		**GISSESS**	
9	Netname	4	**GISLU**	
Connection to a Sun Solaris system The values in this section of the table must match those used in the table for Sun Solaris in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**SOL**	
7	Group name		**EXAMPLE**	
8	Session name		**SOLSESS**	
9	Netname	5	**SOLARLU**	

Table 76. Configuration worksheet for z/VSE using APPC (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to an AS/400 system The values in this section of the table must match those used in the table for AS/400 in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**AS4**	
7	Group name		**EXAMPLE**	
8	Session name		**AS4SESS**	
9	Netname	3	**AS400LU**	
Connection to a z/OS system without CICS The values in this section of the table must match those used in the table for z/OS in the *WebSphere MQ Intercommunication* book, as indicated.				
6	Connection name		**z/OS**	
7	Group name		**EXAMPLE**	
8	Session name		**MVSSESS**	
9	Netname	4	**MVSLU**	

Explanation of terms

1 Network ID
This is the unique ID of the network to which you are connected. Your system administrator will tell you this value.

2 Node name
This is the name of the SSCP which owns the CICS for z/VSE region.

3 Local LU name
This is the unique VTAM APPLID of this CICS for z/VSE region.

4 Transaction Program name
WebSphere MQ applications trying to converse with this queue manager will specify a transaction name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. WebSphere MQ for z/VSE uses a name of MQ01.

5 LAN destination address
This is the LAN destination address that your partner nodes will use to communicate with this host. It is usually the address of the 3745 on the same LAN as the partner node.

6 Connection name
This is a 4-character name by which each connection will be individually known in CICS RDO.

7 Group name
You choose your own 8-character name for this value. Your system may already have a group defined for connections to partner nodes. Your system administrator will give you a value to use.

8 Session name
This is an 8-character name by which each session will be individually known. For clarity we use the connection name, concatenated with 'SESS'.

9 **Netname**
This is the LU name of the WebSphere MQ queue manager on the system with which you are setting up communication.

Establishing an LU 6.2 connection

This example is for a connection to an OS/2 system. The steps are the same whatever platform you are using; change the values as appropriate.

Defining a connection

1. At a CICS command line type:
   `CEDA DEF CONN(`*connection name*`)` **6** `GROUP(`*group name*`)` **7**

   For example:
   `CEDA DEF CONN(OS2) GROUP(EXAMPLE)`
2. Press Enter to define a connection to CICS.

```
 DEF CONN(OS2) GROUP(EXAMPLE)
 OVERTYPE TO MODIFY
  CEDA  DEFine
   Connection     : OS2
   Group          : EXAMPLE
   DEscription  ==>
  CONNECTION IDENTIFIERS
   Netname      ==> OS2LU
   INDsys       ==>
  REMOTE ATTRIBUTES
   REMOTESystem ==>
   REMOTEName   ==>
  CONNECTION PROPERTIES
   ACcessmethod ==> Vtam              Vtam | IRc | INdirect | Xm
   Protocol     ==> Appc              Appc | Lu61
   SInglesess   ==> No                No | Yes
   DAtastream   ==> User              User | 3270 | SCs | STrfield | Lms
   RECordformat ==> U                 U | Vb
  OPERATIONAL PROPERTIES
+  AUtoconnect  ==> Yes               No | Yes | All
  I New group EXAMPLE created.

  DEFINE SUCCESSFUL                          TIME:  16.49.30  DATE: 96.054
PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

3. On the panel change the **Netname** field in the CONNECTION IDENTIFIERS section to be the LU name (**9**) of the target system.
4. In the CONNECTION PROPERTIES section set the **ACcessmethod** field to `Vtam` and the **Protocol** to `Appc`.
5. Press Enter to make the change.

Defining a session

1. At a CICS command line type:
   `CEDA DEF SESS(`*session name*`)` **8** `GROUP(`*group name*`)` **7**

   For example:
   `CEDA DEF SESS(OS2SESS) GROUP(EXAMPLE)`
2. Press Enter to define a session for the connection.

```
 DEF SESS(OS2SESS) GROUP(EXAMPLE)
 OVERTYPE TO MODIFY
  CEDA  DEFine
   Sessions     ==> OS2SESS
   Group        ==> EXAMPLE
   DEscription  ==>
  SESSION IDENTIFIERS
   Connection   ==> OS2
   SESSName     ==>
   NETnameq     ==>
   MOdename     ==> #INTER
  SESSION PROPERTIES
   Protocol     ==> Appc              Appc | Lu61
   MAximum      ==> 008 , 004         0-999
   RECEIVEPfx   ==>
   RECEIVECount ==>                   1-999
   SENDPfx      ==>
   SENDCount    ==>                   1-999
   SENDSize     ==> 04096             1-30720
+  RECEIVESize  ==> 04096             1-30720
   S CONNECTION MUST BE SPECIFIED.

                                        TIME:  14.23.19   DATE:  96.054
 PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

3. In the SESSION IDENTIFIERS section of the panel specify the Connection name (**6**) in the **Connection** field and set the **MOdename** to `#INTER`.
4. In the SESSION PROPERTIES section set the **Protocol** to `Appc` and the **MAximum** field to `008 , 004`.

Installing the new group definition

1. At a CICS command line type:
   `CEDA INS GROUP(`*group name*`)` **7**
2. Press Enter to install the new group definition.

   Note: If this connection group is already in use you may get severe errors reported. If this happens, take the existing connections out of service, retry the above group installation, and then set the connections in service using the following commands:
   a. `CEMT I CONN`
   b. `CEMT S CONN(*) OUTS`
   c. `CEDA INS GROUP(`*group name*`)`
   d. `CEMT S CONN(*) INS`

What next?

The LU 6.2 connection is now established. You are ready to complete the configuration. Go to "WebSphere MQ for z/VSE configuration" on page 996.

Establishing a TCP/IP connection

TCP/IP connections do not require the configuration of additional profiles as does the LU 6.2 protocol. Instead, WebSphere MQ for z/VSE processes the WebSphere MQ listener program during WebSphere MQ startup.

The WebSphere MQ listener program waits for remote TCP/IP connection requests. As these are received, the listener starts the receiver MCA to process the remote

connection. When the remote connection is received from a client program, the receiver MCA starts the WebSphere MQ server program.

Note: There is one WebSphere MQ server process for each client connection.

Provided that the MQ Listener is active and TCP/IP is active in a z/VSE partition, TCP/IP connections can be established.

WebSphere MQ for z/VSE configuration

Configuring WebSphere MQ for z/VSE involves these tasks:
- Configuring channels.
- Defining a local queue.
- Defining a remote queue.
- Defining a sender channel.
- Defining a receiver channel.

Note: WebSphere MQ for z/VSE does not understand the format of the WebSphere MQ channel ping command. The only way to verify your WebSphere MQ definitions is to start the channels and put messages onto remote queues.

Configuring channels

Examples are given for connecting WebSphere MQ for z/VSE and WebSphere MQ for OS/2. If you wish connect to another WebSphere MQ platform use the appropriate set of values from the table in place of those for OS/2.

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

Refer to the sections "Defining a local queue" on page 999 and "Defining a remote queue" on page 1001 for details of how to create queue definitions, and "Defining a SNA LU 6.2 sender channel" on page 1003 and "Defining a SNA LU 6.2 receiver channel" on page 1005 for details of how to create channels.

Table 77. Configuration worksheet for WebSphere MQ for z/VSE

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**VSEP**	
B	Local queue name		**VSE.LOCALQ**	
Connection to WebSphere MQ for OS/2 The values in this section of the table must match those used in the worksheet table for OS/2 in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name	A	**OS2**	
D	Remote queue name		**OS2.REMOTEQ**	
E	Queue name at remote system	B	**OS2.LOCALQ**	
F	Transmission queue name		**OS2**	
G	Sender channel name		**VSE.OS2.SNA**	
I	Receiver channel name	G	**OS2.VSE.SNA**	

Table 77. Configuration worksheet for WebSphere MQ for z/VSE (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to WebSphere MQ for Windows NT The values in this section of the table must match those used in the worksheet table for Windows NT in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender channel name		**VSE.WINNT.SNA**	
I	Receiver channel name	G	**WINNT.VSE.SNA**	
Connection to WebSphere MQ for AIX The values in this section of the table must match those used in the worksheet table for AIX in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender channel name		**VSE.AIX.SNA**	
I	Receiver channel name	G	**AIX.VSE.SNA**	
Connection to WebSphere MQ for HP-UX The values in this section of the table must match those used in the worksheet table for HP-UX in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender channel name		**VSE.HPUX.SNA**	
I	Receiver channel name	G	**HPUX.VSE.SNA**	
Connection to WebSphere MQ for AT&T GIS UNIX The values in this section of the table must match those used in the worksheet table for GIS UNIX in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**GIS**	
D	Remote queue name		**GIS.REMOTEQ**	
E	Queue name at remote system	B	**GIS.LOCALQ**	
F	Transmission queue name		**GIS**	
G	Sender channel name		**VSE.GIS.SNA**	
I	Receiver channel name	G	**GIS.VSE.SNA**	
Connection to WebSphere MQ for Sun Solaris The values in this section of the table must match those used in the worksheet table for Sun Solaris in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender channel name		**VSE.SOLARIS.SNA**	
I	Receiver channel name	G	**SOLARIS.VSE.SNA**	

Table 77. Configuration worksheet for WebSphere MQ for z/VSE (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to WebSphere MQ for AS/400 The values in this section of the table must match those used in the worksheet table for AS/400 in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender channel name		**VSE.AS400.SNA**	
I	Receiver channel name	G	**AS400.VSE.SNA**	
Connection to WebSphere MQ for z/OS without CICS The values in this section of the table must match those used in the worksheet table for z/OS in the *WebSphere MQ Intercommunication* book, as indicated.				
C	Remote queue manager name		**z/OS**	
D	Remote queue name		**MVS™.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**z/OS**	
G	Sender channel name		**VSE.MVS.SNA**	
I	Receiver channel name	G	**MVS.VSE.SNA**	

For TCP/IP, the sender channel name **G** and the receiver channel name **I**, in the preceding table, can be `VSE.sys.tcp` and `sys.VSE.TCP` respectively.

In both cases `sys` represents the remote system name, for example, `OS2`. Therefore, in this case, **G** becomes `VSE.OS2.TCP` and **I** becomes `OS2.VSE.TCP`.

WebSphere MQ for z/VSE sender-channel definitions

```
     Local Queue
            Object Type :   L
            Object Name :   OS2              F
              Usage Mode:   T (Transmission)

     Remote Queue
            Object Type :   R
            Object Name :   OS2.REMOTEQ      D
     Remote QUEUE Name :    OS2.LOCALQ       E
     Remote QM    Name :    OS2              C
     Transmission Name :    OS2              F

     Sender Channel
           Channel name :   VSE.OS2.SNA      G
           Channel type :   S (Sender)
Transmission queue name :   OS2              F
                Partner :   OS2              6
                TP Name :   MQTP
```

WebSphere MQ for z/VSE receiver-channel definitions

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

     Receiver Channel
           Channel name :   OS2.VSE.SNA      I
           Channel type :   R (Receiver)
```

Defining a local queue

1. Run the WebSphere MQ master terminal transaction MQMT.

```
01/03/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
14:51:52           *** Master Terminal Main Menu ***                CIC1
MQWMTP                                                              A004
                          SYSTEM IS ACTIVE

                            1. Configuration

                            2. Operations

                            3. Monitoring

                            4. Browse Queue Records


                         Option: 1







Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Clear/PF3=Exit                            Enter=Select
```

2. Select option 1 to configure.

```
12/24/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
10:56:35           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A000
                           SYSTEM IS ACTIVE

                          Maintenance Options :
                              1. Global System Definition
                              2. Queue Definitions
                              3. Channel Definitions
                              4. Code Page Definitions
                              5. Namelist Definitions

                          Display Options     :
                              6. Global System Definition
                              7. Queue Definitions
                              8. Channel Definitions
                              9. Code Page Definitions
                             10. Namelist Definitions

                       Option:



Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Process  PF2=Return  PF3=Exit
```

3. Select option 2 to work with queue definitions.

```
01/03/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
14:53:16                Queue Main Options                           CIC1
MQWMQUE                                                              A004
                            SYSTEM IS ACTIVE

       Default Q Manager. : VSE.TS.QM1

       Object Type. . . . : L      L = Local Queue
                                   R = Remote Queue
                                   AQ = Alias Queue
                                   AM = Alias Queue Manager
                                   AR = Alias Reply Queue

       Object Name. . . . : VSE.LOCALQ









PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

4. Select an Object type of L and specify the name of the queue.
5. Press PF5.

```
11/19/2009        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
14:24:40               Queue Definition Record                       CIC1
MQWMQUE                 Local Queue Definition                       A000

Object Name  . . . . . . . : VSE.LOCALQ
Description line 1 . . . : Test Q
Description line 2 . . . :

Dual Update Queue  . . . :

Put Enabled  . . . . . . . : Y    Inbound status . . : A
Get Enabled  . . . . . . . : Y    Outbound status  . : A

Accounting, Statistics & Monitoring
Queue accounting . . . . : Q    Queue monitoring . : Q
Queue statistics . . . . : Q

Automatic Reorganization
Reorganize . . . . . . . . : N    Start Time: 0000 Interval: 0000
VSAM Catalog . . . . . . . :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF10=Queue  PF12=Delete
```

6. Press PF5 again.

```
11/11/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
13:00:08                Queue Extended Definition                    CIC1
MQMMQUE                                                              A005

Object Name: VSE.LOCALQ

General               Maximums                   Events
Type   . . : Local    Max. Q depth . : 00001000  Service int. event: N
File name  : MQFO0001  Max. msg length: 00004000  Service interval  : 00000000
Usage  . . : N        Max. Q users . : 00000100  Max. depth event  : N
Shareable  : Y        Max. gbl locks : 00000200  High depth event  : N
                      Max. lcl locks : 00000200  High depth limit  : 000
                                                 Low depth event . : N
Triggering                                       Low depth limit . : 000
Enabled  . : N        Transaction id.:
Type . . . :          Program id . . :
Max. starts: 0001     Terminal id  . :
Restart  . : N        Channel name . :
User data  :
           :

Requested record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
            PF9=List  PF10=Queue
```

7. Specify the name of a CICS file to store messages for this queue.
8. If you are creating a transmission queue, specify a **Usage Mode** of T, a **Program ID** of MQPSEND, and a **Channel Name**< G >.

   For a normal queue specify a **Usage Mode** of N.
9. Press PF5 again.

Defining a remote queue

1. Run the WebSphere MQ master terminal transaction MQMT.

```
01/03/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
14:51:52           *** Master Terminal Main Menu ***                CIC1
MQWMTP                                                              A004
                         SYSTEM IS ACTIVE

                           1. Configuration

                           2. Operations

                           3. Monitoring

                           4. Browse Queue Records


                       Option: 1









Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Clear/PF3=Exit                          Enter=Select
```

2. Select option 1 to configure.

```
12/24/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
10:56:35           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A000
                          SYSTEM IS ACTIVE

                         Maintenance Options :
                             1. Global System Definition
                             2. Queue Definitions
                             3. Channel Definitions
                             4. Code Page Definitions
                             5. Namelist Definitions

                         Display Options     :
                             6. Global System Definition
                             7. Queue Definitions
                             8. Channel Definitions
                             9. Code Page Definitions
                            10. Namelist Definitions

                      Option:



Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Process  PF2=Return  PF3=Exit
```

3. Select option 2 to work with queue definitions.

```
01/03/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
15:02:59                Queue Main Options                           CIC1
MQWMQUE                                                             A004
                             SYSTEM IS ACTIVE

       Default Q Manager. : VSE.TS.QM1

       Object Type. . . . : R      L = Local Queue
                                   R = Remote Queue
                                   AQ = Alias Queue
                                   AM = Alias Queue Manager
                                   AR = Alias Reply Queue

       Object Name. . . . : OS2.REMOTEQ









PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

4. Select an **Object type** of **R** and specify the name of the queue.
5. Press PF5.

```
01/03/2008         IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
15:04:25                Queue Definition  Record                     CIC1
MQWMQUE           QM - VSE.TS.QM1                                    A004

                       Remote Queue Definition

Object Name. . . . . . . . : OS2.REMOTEQ
Description line 1 . . . . : Test remote queue on OS/2
Description line 2 . . . . :

Put Enabled  . . . . . . . : Y   Y=Yes, N=No
Get Enabled  . . . . . . . : Y   Y=Yes, N=No

Remote Queue Name. . . . . : OS2.LOCALQ
Remote Queue Manager Name. : OS2
Transmission Queue Name. . : OS2




Record being added   - Press ADD key again.
PF2=Return  PF3=Quit  PF4/Enter=Read   PF5=Add  PF6=Update
                                       PF9=List PF12=Delete
```

6. Specify a remote queue name, remote queue manager name, and transmission queue name.
7. Press PF5.

Defining a SNA LU 6.2 sender channel

1. Run the WebSphere MQ master terminal transaction MQMT.

```
01/03/2008       IBM WebSphere MQ for z/VSE Version 3.0.0            TSMQ300
14:51:52           *** Master Terminal Main Menu ***                 CIC1
MQWMTP                                                               A004
                         SYSTEM IS ACTIVE

                           1. Configuration

                           2. Operations

                           3. Monitoring

                           4. Browse Queue Records


                    Option: 1




Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Clear/PF3=Exit                            Enter=Select
```

2. Select option 1 to configure.

```
12/24/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
10:56:35           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A000
                          SYSTEM IS ACTIVE

                         Maintenance Options :
                             1. Global System Definition
                             2. Queue Definitions
                             3. Channel Definitions
                             4. Code Page Definitions
                             5. Namelist Definitions

                         Display Options     :
                             6. Global System Definition
                             7. Queue Definitions
                             8. Channel Definitions
                             9. Code Page Definitions
                            10. Namelist Definitions

                       Option:



Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Process  PF2=Return  PF3=Exit
```

3. Select option 3 to work with channel definitions.

```
11/19/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:27:20                    Channel Record           DISPLAY        CIC1
MQWMCHN                                                             A000
Channel  : VSE1.SNA.AIX1
 Desc. . :
 Protocol: L (L/T)  Type : S (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000003
 Get retry number . . . . . : 00000001   Short retry interval  . . : 000000003
 Get retry delay (secs) . . : 00000010   Long retry interval . . . : 000000010
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000020000
 Transmission queue name. . : VSE1.AIX1.XQ1
 TP name. . : MQ01

Sender/Receiver/Server/Requester
 Connection : AIX1
 Max Messages per Batch . . : 000050     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 032000     Split Msg(Y/N)  . . . . . : N
 Max TCP/IP Wait  . . . . . : 000000     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

4. Complete the parameter fields as indicated, specifically the fields **Channel name**< **G** >, **Channel type**, **Connection ID**, **Remote task ID**, and **Transmit queue name**< **F** >.

   All other parameters can be entered as shown.
5. Press PF5.

Defining a SNA LU 6.2 receiver channel

1. Run the WebSphere MQ master terminal transaction MQMT.

```
01/03/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
14:51:52           *** Master Terminal Main Menu ***                CIC1
MQWMTP                                                              A004
                          SYSTEM IS ACTIVE

                            1. Configuration

                            2. Operations

                            3. Monitoring

                            4. Browse Queue Records


                         Option: 1







Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Clear/PF3=Exit                            Enter=Select
```

2. Select option 1 to configure.

```
12/24/2008       IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
10:56:35           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A000
                           SYSTEM IS ACTIVE

                          Maintenance Options :
                              1. Global System Definition
                              2. Queue Definitions
                              3. Channel Definitions
                              4. Code Page Definitions
                              5. Namelist Definitions

                          Display Options     :
                              6. Global System Definition
                              7. Queue Definitions
                              8. Channel Definitions
                              9. Code Page Definitions
                             10. Namelist Definitions

                       Option:



Please enter one of the options listed.
    5655-U97 Copyright IBM Corp. 2008. All Rights Reserved.
Enter=Process  PF2=Return  PF3=Exit
```

3. Select option 3 to work with channel definitions.

```
11/19/2009        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQBD
14:27:20                    Channel Record           DISPLAY        CIC1
MQWMCHN                                                             A000
Channel  : AIX1.SBA.VSE1
 Desc. . :
 Protocol: L (L/T)  Type : R (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 00000      Short/Long retry count  . : 000000003
 Get retry number . . . . . : 00000000   Short retry interval  . . : 000000003
 Get retry delay (secs) . . : 00000000   Long retry interval . . . : 000000010
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000020000
 Transmission queue name. . :
 TP name. . :

Sender/Receiver/Server/Requester
 Connection :
 Max Messages per Batch . . : 000050     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 032000     Split Msg(Y/N)  . . . . . : N
 Max TCP/IP Wait  . . . . . : 000000     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

4. Complete the parameter fields as indicated, specifically the field **Channel name**< **L** >.

   All other parameters can be entered as shown.
5. Press PF5.

Defining a TCP/IP sender channel

To define a TCP/IP sender channel, carry out the following procedure:

1. Run the WebSphere MQ master terminal transaction MQMT.
2. Select option 1 to configure.
3. Select option 3 to work with channel definitions. The screen shown in Figure 88 on page 1007 is displayed:

```
11/19/2009        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQBD
14:27:20                    Channel Record           DISPLAY         CIC1
MQWMCHN                                                              A000
Channel  : VSE1.TCP.NT1
 Desc. . :
 Protocol: T (L/T)  Type : S (S=Snd/R=Rcv/V=Srv/Q=Req/C=svrConn)  Enabled : Y

Sender/Server
 Remote TCP/IP port . . . . : 01414      Short/Long retry count  . : 000000003
 Get retry number . . . . . : 00000003   Short retry interval  . . : 000000003
 Get retry delay (secs) . . : 00000030   Long retry interval . . . : 000000010
 Convert msgs(Y/N). . . . . : N          Batch interval  . . . . . : 000020000
 Transmission queue name. . : VSE1.NT1.XQ1
 TP name. . :

Sender/Receiver/Server/Requester
 Connection : NT1SERV
 Max Messages per Batch . . : 000050     Message Sequence Wrap . . : 999999999
 Max Message Size . . . . . : 0004096    Dead letter store(Y/N)  . : N
 Max Transmission Size  . . : 065535     Split Msg(Y/N)  . . . . . : N
 Max TCP/IP Wait  . . . . . : 000000     Channel statistics  . . . : Q
                                         Channel monitoring  . . . : Q
Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF11=Ext PF12=Del
```

Figure 88. Channel configuration panel

4. Complete the parameter fields as follows:
   - Channel name - **G** on the configuration worksheet.
   - Partner - should contain either the domain name or the IP address of the remote host, for example NTSERV1 or 1.20.33.44.
   - Port - the port number must match the port number configured for the remote host. This is configured in the global system definition of the remote host. The default port number for WebSphere MQ for z/VSE is 1414.
   - Transmission queue name - **F** on the configuration worksheet.
   - Protocol - enter `T` for TCP/IP.
   - Channel type - enter `S` for sender.

   Note:

   a. The TP Name is not used by TCP/IP channels.

   b. Ensure that the parameter field values match the values of the receiver channel definition of the same name on the remote host.
5. Press PF5 (Add) to add the new channel definition.

Defining a TCP/IP receiver channel

To define a TCP/IP receiver channel, carry out the following procedure:

1. Run the WebSphere MQ master terminal transaction MQMT.
2. Select option 1 to configure.
3. Select option 3 to work with channel definitions. The screen shown in Figure 88 is displayed.
4. Complete the parameter fields as follows:
   - Channel name - **G** on the configuration worksheet.
   - Protocol - enter `T` for TCP/IP.
   - Channel type - enter `R` for receiver.

Note:

a. The Partner and Port are not required for a TCP/IP receiver channel.
b. The TP Name is not used by TCP/IP channels.
c. Ensure that the parameter field values match the values of the sender channel definition of the same name on the remote host.

5. Press PF5 (Add) to add the new channel definition.

Appendix F. WebSphere MQ server

The WebSphere MQ for z/VSE server is a special instance of a Message Channel Agent (MCA) that accepts and manages client MQI connections. WebSphere MQ for z/VSE accepts client connections using TCP/IP only.

Client applications, that is those application running in an WebSphere MQ client environment, connect to a server instance using an MQCONN or MQCONNX MQI call. Once successfully connected, the client application issues MQI calls which are performed by the WebSphere MQ server, and the results of the call are reported to the client application. The client/server session ends when the client applications uses the MQDISC call.

The WebSphere MQ server is provided with the WebSphere MQ for z/VSE product. For this feature to be available, the MQ TCP/IP listener task must be running in the CICS region that hosts the queue manager. When the listener task detects a connection request from a client application it starts a server instance to negotiate the connection and service the client application's MQI calls.

Because the WebSphere MQ server runs as an MCA instance, it runs as transaction MQ01 in the CICS region.

For more information about the WebSphere MQ client, see Chapter 10, "WebSphere MQ clients," on page 623, and *WebSphere MQ Clients*, (GC34-6590).

Server MQI support

The WebSphere MQ for z/VSE server supports clients issuing the following MQI calls:

MQCONN
: Connect queue manager

MQCONNX
: Connect queue manager (extended)

MQOPEN
: Open message queue

MQGET
: Get message

MQPUT
: Put message

MQPUT1
: Put one message

MQINQ
: Inquire about object attributes

MQSET
: Set object attributes

MQCLOSE
: Close object

MQDISC
: Disconnect queue manager

MQCMIT
: Commit changes

MQBACK
: Backout changes

MQBUFMH
Buffer to message handle
MQCRTMH
Create message handle
MQDLTMH
Delete message handle
MQDLTMP
Delete message property
MQINQMP
Inquire message property
MQMHBUF
Message handle to buffer
MQSETMP
Set message property

The MQI calls, generally, expect parameters that specify particular options relevant to the MQI call. In some cases parameters are data structure that indicate a version. Each WebSphere MQ server, by platform, potentially supports different options and different data structure versions.

The MQI options and data structure versions supported by the WebSphere MQ for z/VSE server are the same as those supported by z/VSE applications local to the VSE queue manager. For more specific details, see Appendix B, "Application Programming Reference," on page 717.

Security considerations

In the WebSphere MQ for z/VSE environment, there are several ways to implement security for WebSphere MQ client access through an WebSphere MQ server. These include:
- WebSphere MQ queue manager security.
- Channel exits.

Security can also be implemented by combining both queue manager security and channel exits.

Queue manager security

To use queue manager security to authenticate and control WebSphere MQ client access, it is necessary to install the z/VSE queue manager with the security feature active. This is described in "Installing security" on page 19.

The client must identify itself with both a userid and password using the WebSphere MQ client environment variables: MQ_USER_ID and MQ_PASSWORD. The WebSphere MQ for z/VSE server uses these values to authenticate the client and ensure that access to queue manager resources is controlled by the permission granted to the authenticated user.

User permissions should be granted as if the user were a user using local queue manager resources. In addition, the user that starts the queue manager's MQ listener task must be a surrogate for all client users. The user that starts the MQ listener is generally the queue manager startup user.

Some WebSphere MQ clients do not support client authentication using these variables. If the client system does not support this type of authentication, then channel security exits are required to authenticate the client.

Channel exits

Channel security exits can be used to authenticate WebSphere MQ client connections. If a security exit is defined to the server-connection channel, it is executed during when the client has established a connection to the client, but before the MQCONN or MQCONNX call is deemed successful.

A security exit can also be defined to the client-connection channel. If an exit is defined, it is passed data generated by server's security exit. If no data is generated by the server's security exit, the client's security exit is still executed. The two security exits can exchange data, or send "null" data flows, until both exits either accept or reject the connection.

If security exits are to be used in conjunction with queue manager security, the security exit can set the MQCD data structure (passed to the exit) to identify a user and password. Relevant fields are:

MCAUserIdentifier
RemotePassword

If queue manager security is active, and the MQ_USER_ID and MQ_PASSWORD fields are not used by the client, the WebSphere MQ for z/VSE server authenticates the client user using these fields. If the authentication is successful, the permissions granted to the user are used to control access to queue manager resources.

Send and receive exits can also be used for security purposes. For example, the client might encrypt data before it is sent to the server, and a receive exit defined to the server-connection channel might decrypt the data before it is processed by the server.

Code page conversion

There is no guarantee that the code page of the WebSphere MQ client will match that of the WebSphere MQ server. In addition, message data retrieved by the server for the client may use a different code page to the one used in the client environment. For these reasons it may be necessary for the server to manage code page conversion.

The WebSphere MQ for z/VSE server uses Language Environment/VSE services to manage code page conversion. This means the client code page and the server code page must represent a pair that Language Environment/VSE is capable of converting.

Language Environment/VSE is shipped with a number of default conversion tables. These are documented in the *Language Environment V1R4 C Run-Time Programming Guide*, SC33-6688.

Client programs are not limited to the default code page conversion tables supplied with Language Environment/VSE. It is possible to create your own conversion tables using the Language Environment/VSE code page conversion utilities.

Creating code page conversion tables

The code page conversion utilities can be reviewed in the Language Environment C Run-Time Programming Guide, SC33-6688. For convenience, an overview of the steps needed to create a code page conversion table are provided here.

Here are the items involved in the creation of Language Environment/VSE code page conversion tables:
- "Code page numbers"
- "Code page translation tables"
- "The GENXLT utility" on page 1013
- "CSD definitions" on page 1013

If you follow the suggestions set out for each item, you can create your own code page conversion tables. This process is only necessary if you have an MQ client that uses a code page that does not have a default translation table to and from your z/VSE server code page.

Code page numbers

Code pages have their own unique number; for example, 037 for USA and Canada, 273 for Germany and Austria, and 285 for the United Kingdom. You should determine the numbers of the code pages you want to convert. Some code pages have names that are not numeric, but these generally will have numeric aliases. For example, code page 819 is an alias for code page ISO8859-1. If you want to use code page ISO8859-1, the WebSphere MQ client should set its code page to 819.

There are several ways to do this. One is to set the MQCCSID environment variable to 819. For example:

```
c:\>set MQCCSID=819
```

If your client program uses Java classes, you can set the CCSID attribute of the MQEnvironment object. For example:

```
MQEnvironment.CCSID = 819;
```

The Language Environment/VSE source file EDCUCSNM.A, in PRD2.SCEEBASE, documents some of the code page aliases. You can create your own aliases.

Code page translation tables

Code page translation tables have a strict naming convention. This is because the appropriate translation table name is built dynamically by Language Environment/VSE based on the two code page numbers involved in the translation.

Translation tables follow the naming convention EDCUfftt, where ff is a two character synonym for the 'from' code page, and tt is a two character synonym for the 'to' code page. For example, the two character synonym for code page 37 is EA and the synonym for code page 273 is EB. Consequently, the translation table name for 37 to 273 is EDCUEAEB and the translation table for 273 to 37 is EDCUEBEA.

The two character synonyms for code page numbers are documented in the Language Environment C Run-Time Programming Guide, SC33-6688. They are also coded in a phase used by Language Environment/VSE to dynamically build the translation table name from the code page numbers. This phase is EDCUCSNM and it is used by Language Environment/VSE when code page conversion services are employed.

When these services are used, as they are by the WebSphere MQ for z/VSE server, only the code page numbers are identified. Language Environment/VSE uses the EDCUCSNM to dynamically build the name of the translation table that is subsequently used for code page conversion.

The following is an example of EDCUCSNM source entries:

```
EDCCSNAM TYPE=ENTRY,CODESET='IBM-037',CODE='EA'
EDCCSNAM TYPE=ENTRY,CODESET='IBM-273',CODE='EB'
EDCCSNAM TYPE=ENTRY,CODESET='IBM-274',CODE='EC'
EDCCSNAM TYPE=ENTRY,CODESET='IBM-275',CODE='ED'
EDCCSNAM TYPE=ENTRY,CODESET='IBM-277',CODE='EE'
EDCCSNAM TYPE=ENTRY,CODESET='IBM-278',CODE='EF'
```

The source for EDCUCSNM is found in PRD2.SCEEBASE library, EDCUCSNM.A file.

Code page translation source files are also found in the PRD2.SCEEBASE library. These source files are prefixed EDCU and have an 'X' extension, for example, EDCUAAEY.X. These can be used as a basis for new conversion tables.

The source files contain three columns. The first two columns contain hexadecimal values from 0x00 to 0xFF; the third column documents the relevant character being converted. For example, (showing only the first 16 entries):

```
0x00 0x00 <NUL>
0x01 0x01 <SOH>
0x02 0x02 <STX>
0x03 0x03 <ETX>
0x04 0xdc <SEL>
0x05 0x09 <tab>
0x06 0xc3 <RNL>
0x07 0x1c <DEL>
0x08 0xca <GE>
0x09 0xb2 <SPS>
0x0a 0xd5 <RPT>
0x0b 0x0b <vertical-tab>
0x0c 0x0c <form-feed>
0x0d 0x0d <carriage-return>
0x0e 0x0e <SO>
0x0f 0x0f <SI>
```

The GENXLT utility

The source files are assembled using the Language Environemt/VSE GENXLT utility and then link edited to produce executable phases. Each phase represents a conversion from one code page to another. They are mono-directional, so there should be two phases for each translation needed.

The following example shows JCL that could be used to generate a translation table phase using the GENXLT utility.

```
// JOB MQGENXLT
// LIBDEF *,SEARCH=PRD2.SCEEBASE
// LIBDEF PHASE,CATALOG=target.sublib
// OPTION LINK,CATAL
// EXEC EDCGNXLT,PARM='IFILE(DD:PRD2.SCEEBASE(EDCU1AEA.X)),NODBCS,     X
               NAME(EDCU1AEA)'
/*
// EXEC LNKEDT
/*
/&
```

The code page name table, EDCUCSNM, does not require the GENXLT utility. The source can be punched from the Language Environment/VSE installation sublibrary, modified as appropriate, assembled and link edited.

CSD definitions

Since WebSphere MQ for z/VSE runs under CICS, the EDCUCSNM and the translation table phases must be known to CICS. Consequently, you will need CSD

program entries for these phases. The language type is assembler, and the CSD group for default tables is CEE. You can use the CEDA transaction to look at existing entries. For example:

```
OBJECT CHARACTERISTICS
 CEDA View
  PROGram        : EDCUAAEY
  Group          : CEE
  Language       : Assembler        CObol | Assembler | C | Pli | Rpg
  RELoad         : No               No | Yes
  RESident       : No               No | Yes
  RSl            : 00               0-24 | Public
  Status         : Enabled          Enabled | Disabled
 REMOTE ATTRIBUTES
  REMOTESystem   :
  REMOTEName     :
  Transid        :
  Executionset   : Fullapi          Fullapi | Dplsubset
```

Appendix G. System messages

This appendix describes the messages issued by WebSphere MQ.

WebSphere MQ generates both internal and external messages. Internal messages are generated when an application program activates WebSphere MQ and an abnormal condition occurs.

These messages are stored on the system log queue when it is available; otherwise, the CICS CSMT Transient Data (TD) queue is used.

API system messages

These messages consist of five lines of text, each with a maximum of 78 characters, together with two lines of error code information as follows:

Line 1 -

```
MQInnnnnns PRG:pppppppp TRN:tttt TRM:rrrr TSK:ccccc mm/dd/yy hh:mm:ss
```

Where:

nnnnnn WebSphere MQ message code - see "WebSphere MQ message codes" on page 1016

s Severity

Severity values have the following meanings:

- **I** An information message. No error has occurred.
- **W** A warning message. A condition has been detected of which you should be aware. You may need to take further action.
- **E** An error message. An error has been detected that the system typically corrects. However, you may have to intervene.
- **C** A critical error message. An error has been detected that may severely affect user or system operation. This requires your immediate intervention.

pppppppp
CICS Program name

tttt CICS Transaction code

rrrr CICS Term ID

ccccc CICS Task ID

mm/dd/yy
Date

hh:mm:ss
Time

Line 2 - Textual description of message

Line 3 - Queue name, if available

Line 4 - Channel name, if available

Line 5 - Detail of message (optional)

Line 6 -

```
EIBFN:fff EIBRCODE:rrrrrrrrrrrr EXEC LINE: lllll
```

Where:
fff EIBFN value at time of condition
rrrrrrrr
EIBRCODE
11111 The DEBUG CICS command number

Line 7 -

```
EIBRESP: rrrrrrrr EIBRESP2: ssssssss EIBRSRCE:cccccccc ABCODE: aaaa
```

Where:
rrrrrrrr
EIBRESP
ssssssss
EIBRESP2
cccccccc
EIBRSRCE
aaaa CICS ABENDCODE

WebSphere MQ message definitions

Each WebSphere MQ message provides the following information:

Explanation:
This section explains what the message or code means, why it occurred, and what caused it.

Function
This section indicates which modules issued the message, to assist in diagnosing problems.

Operator action
If an operator response is necessary, this section describes what the appropriate responses are, and what their effect is. If this information is omitted, no operator response is required.

System action
This part describes what is happening as a result of the condition causing the message or code. If this information is omitted, no system action is taken.

WebSphere MQ messages

WebSphere MQ system messages are numbered 000000 through 900000, and they are listed in this book in numeric order. However, not all numbers have been used, therefore, the list is not continuous.

WebSphere MQ console messages are numbered from MQI0001 onwards, and they are listed in this book in numeric order; see “Console Messages” on page 1052.

WebSphere MQ message codes

000000I Queue manager started

Explanation: The local queue manager has been started.

System action: The queue manager is available for queuing services.

User response: None.

000001I Queue manager stop requested

Explanation: A request to stop the queue manager has been issued.

System action: Queue manager services are stopping.

User response: None.

000002I Queue manager stopped

Explanation: The local queue manager has been stopped.

System action: Queue manager services are unavailable until the system is restarted.

User response: None.

000003E Channel Message Sequence Number error

Explanation: The received MSN does not match the expected MSN.

System action: Fatal error - Communication is terminated.

User response:

1. Review the LOCAL MSN and the REMOTE MSN in the detail portion of the message.
2. Identify the cause (proper running should preclude this occurrence).
3. Reset the appropriate MSN so that the sender and receiver channel MSN's are equal.
4. Restart communication.

000004W Synch MSG duplicate

Explanation: The received message may be duplicated.

System action: Continue on negotiating.

User response: None.

000007I LU62 session started

Explanation: A communication session was started by MQPRECV.

System action: None.

User response: None.

000009E Channel wrap value negotiation mismatch

Explanation: A sent WRAP value did not match that of the of the receiver.

System action: Channel communication ends.

User response: Review the SENT and RECEIVED match values and change one side to match the other.

000010E LU62 FREE error

Explanation:

For Program MQPRECV

Upon completion of a RECEIVE command the EIBFREE and the EIBERR fields are both not equal to low values.

For Program MQPSEND

As a Server upon completion of a RECEIVE command at least one of EIBERR, EIBRECV and EIBFREE does not equal to low values.

As a Server or Sender upon receipt of an acknowledgement of messages sent the EIBFREE is not equal to low values and the EIBERR is equal to low values.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000011E LU62 EIB error

Explanation: For Program MQPSEND

1. As a Server upon completion of a RECEIVE the EIBERR not equal to low values.
2. As a Server or Sender upon receipt of an acknowledgement of messages sent, the EIBERR is not equal to low values.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000012E LU62 STAT error For Program MQPSEND -

Explanation: As a Server or Sender upon receipt of an acknowledgement of messages sent, the EIBRECV is not equal to low values.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000013E LU62 ALLOC error

Explanation: For Program MQPSEND

- As a Sender upon completion of an ALLOCATE command, EIBRCODE is not equal to low values and all retries have been performed.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000014W LU62 ALLOC RETRY error For Program MQPSEND -

Explanation: As a Sender upon completion of an ALLOCATE command, EIBRCODE is not equal to low values and all retry attempts have not been performed.

System action: Non-Fatal error - Allocation is retried until allocation is successful or the retry count equals zero.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.

000015E LU62 CONN error

Explanation: For Program MQPSEND

- As a Sender upon completion of a CONNECT PROCESS command, EIBRCODE is not equal to low values.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000016E LU62 SEND error

Explanation: For Program MQPSEND

- As a Sender or Server upon completion of a SEND command, EIBRCODE is not equal to low values.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

000017I Remote deallocation or communication shutdown

Explanation: For Program MQPSEND or MQPRECV

- The remote MCA deallocated the conversation or closed an active IP socket. Alternatively, a request to shutdown the MQ communications subsystem was issued (via MQ shutdown).

System action: Communication is terminated.

User response: Normally, this is an informational message and no additional user action is required. However, in the event that the deallocation has occurred due to a remote system failure, intervention may be required on the remote system.

000023E Invalid error data

Explanation: For Program MQPSEND or MQPRECV

- The Sender or Receiver MCA received an error data transmission that contained an unrecognized error code.

System action: Fatal error - Communication is terminated.

User response: Review System Log or error TD Queue for messages prior to this message. Proper running should preclude this occurrence.

000024E Invalid MSN value received during negotiation

Explanation: (Reserved)

000025E Fatal response type

Explanation: (Reserved)

000026E Recoverable response type

Explanation: (Reserved)

000029E Parser MSN error

Explanation: (Reserved)

000030E Parser type error

Explanation: (Reserved)

000031E Parser PDM error

Explanation: (Reserved)

000032E Parser SID error

Explanation: (Reserved)

000033E Parser PN error

Explanation: (Reserved)

000034E Parser KEY error

Explanation: (Reserved)

000035E Parser APID error

Explanation: (Reserved)

000038E Parser ORG DT error

Explanation: (Reserved)

000039E Parser ORIG MSN error

Explanation: (Reserved)

000040E Parser BODY error

Explanation: (Reserved)

000041E Parser status error

Explanation: The received message does not have the proper status value.

System action: Fatal error - Communication is terminated.

User response: Review System Log or error TD Queue for messages prior to this message. Proper running should preclude this occurrence. Investigate sender process for programming error.

000042E Parser length error

Explanation: The received message does not have the proper length value.

System action: Fatal error - Communication is terminated.

User response: Review System Log or error TD Queue for messages prior to this message. Proper running should preclude this occurrence. Investigate sender process for programming error.

000051E Queue connection error

Explanation: The QM cannot be connected to.

System action: Fatal error - Communication is terminated.

User response: Review System Log for associated error messages. Ensure that the queue manager has not been shutdown during communication activity.

000052E Queue open error

Explanation: The Sender or Receiver MCA could not open a source or target queue.

System action: Fatal error - Communication is terminated.

User response:

1. For a sender channel, check that the associated transmission queue is valid and enabled. For a receiver channel, check that the intended target queue and the system dead letter queue are valid and enabled.
2. Review the fields in this error message:
   QUEUE ID
   Transmission queue name that failed.
   CHANNEL ID
   Channel name that was connected. This channel identifies the corresponding transmission queue.
   Last line of error message
   Reason code returned from queuer and corresponding description.
3. Correct problem and restart communication.

000053E Queue GET error

Explanation: The Server or Sender could not get a message from the associated transmission queue even if there are messages in the transmission queue.

System action: Fatal error - Communication is terminated.

User response:

1. Review the following fields in the error message:
   QUEUE ID
   Transmission queue name that failed.
   CHANNEL ID
   Channel name that was connected. This channel identifies the corresponding transmission queue.
   Last line of error message
   Reason code returned from queuer and corresponding description.
2. Correct problem and restart communication.

000054E Queue PUT error

Explanation: The RECEIVER could not put a message to an application queue.

System action: There are two possible system actions based upon reason code returned from queuer:
Reason code equals MQRC-Q-FULL or MQRC-Q-SPACE-NOT-AVAILABLE
Non-Fatal error - communication will proceed normally after first putting failed put message on dead letter queue.
All other Reason codes
Fatal error - Communication is terminated.

User response:

1. Review the following fields in the error message:
   QUEUE ID
   Application queue name that failed.
   CHANNEL ID
   Channel name that was connected.
   Last line of error message
   Reason code returned from queuer and corresponding description.
2. User action is based upon returned reason code:
   Reason code equals MQRC-Q-FULL (2053) or MQRC-Q-SPACE-NOT-AVAILABLE (2056)
   Destination application queue was full and the message was placed on the dead letter queue. Determine if destination queue should be expanded to accommodate more messages or an alternate destination used.
   All other Reason codes
   Correct problem and restart communication.

000055E Queue PUT1 error

Explanation: The RECEIVER could not put a message to the dead letter queue.

System action: Fatal error - Communication is terminated.

User response:

1. Review the following fields in the error message:
   QUEUE ID
   The dead letter queue name that failed.
   CHANNEL ID
   Channel name that was connected.
   Last line of error message
   Reason code returned from queuer and corresponding description.
2. Correct problem and restart communication.

000056W Queue CLOSE error

Explanation: The RECEIVER could not close an application queue.

System action: Non-Fatal error - communication will proceed normally. (The unclosed resources, however, will result in a “garbage collection” mechanism be triggered at a proper time to close the unclosed resources).

User response:

1. Review the following fields in the error message:
   QUEUE ID
   Application queue name that failed.
   CHANNEL ID
   Channel name that was connected.
   Last line of error message
   Reason code returned from queuer and corresponding description.
2. Investigate problem

000057E Queue DISC error

Explanation: An error has occurred to DISCONNECT the connecting Queue Manager.

System action: Fatal error - Communication is terminated.

User response: Review System Log or error TD Queue for messages prior to this message. Proper running should preclude this occurrence. Investigate sender process for program error.

000060E Unexpected queue error

Explanation: An unexpected error occurred during queue processing.

System action: Fatal error - queue processing is terminated.

User response: This can occur if the queue manager is shutdown while queue manager connections are active.

000080E Receiver return LON status

Explanation: (Reserved)

000081E Receiver return LON type

Explanation: (Reserved)

000091E SIDRC return format

Explanation: (Reserved)

000100I Function started

Explanation: The requested function has been started

System action: Function is started

User response: None.

000110I Queue modification requested

Explanation: A request was issued to modify a queue.

System action: If the request is successful, the queue is modified.

User response: None.

000112I Queue message modification requested

Explanation: A request was issued to modify messages on a queue.

System action: If the request is successful, the queue messages are modified.

User response: None.

000114I Channel modification requested

Explanation: A request was issued to modify a channel.

System action: If the request is successful, the channel is modified.

User response: None.

000116I Queue manager modification requested

Explanation: A request was issued to modify the queue manager.

System action: If the request is successful, the queue manager is modified.

User response: None.

000700I IBM WebSphere MQ for z/VSE - CICS bridge started

Explanation: The IBM WebSphere MQ for z/VSE CICS bridge has started.

System action: WMQ CICS Bridge started.

User response: None.

000701E StrucId invalid in bridge global data area

Explanation: A bridge task found that the area pointed to by the global data address passed to it in its start data did not contain the expected identifier. This is probably because the monitor task has terminated with a bridge task start request queued. The bridge task checks the global data area at startup and terminates if the structure identifier is not valid.

System action: The request will be processed when the monitor is restarted.

User response: None.

000702I WMQ CICS Bridge Monitor initialization complete

Explanation: Monitor initialization completed successfully.

System action: Bridge monitor ready to process bridge requests.

User response: None.

000704E WMQ CICS Bridge EXEC CICS call error

Explanation: An error occurred in a CICS call issued by the bridge.

System action: Depending on the failure, the bridge related task may proceed or issue an abend.

User response: Use CICS trace facilities to identify the cause on the failure and refer to the CICS Application Programming manual for an explanation. Take appropriate action.

000705E WMQ CICS Bridge start parameter is invalid

Explanation: A parameter provided to the MQ CICS bridge at startup is invalid.

System action: WMQ CICS bridge is terminated.

User response: Correct the parameter and restart the bridge monitor.

000706E WMQ CICS Bridge authentication option invalid

Explanation: The authentication option requested is not supported.

System action: WMQ CICS bridge is terminated.

User response: Choose a supported authentication option for the release of CICS and restart the bridge monitor.

000707I WMQ CICS Bridge not supported on non-z/VSE platforms

Explanation: The bridge is being run on a system other than z/VSE This might work, but is not supported.

System action: None.

User response: None.

000708E WMQ CICS Bridge Monitor must run at a terminal for AUTH option

Explanation: AUTH=VERIFY_UOW was requested. AUTH=VERIFY_UOW on CICS TS 1.1 requires that the monitor is run at a terminal.

System action: Bridge monitor terminated.

User response: Restart the bridge monitor from a terminal or set AUTH=LOCAL.

000709E WMQ CICS Bridge preset security not valid for AUTH=VERIFY_UOW

Explanation: AUTH=VERIFY_UOW was requested. AUTH=VERIFY_UOW on CICS TS 1.1 requires that the monitor is run at a terminal, but that terminal might not have preset security.

System action: Bridge monitor terminated.

User response: Redefine the terminal, or use a different one, before restarting the monitor, or set AUTH=LOCAL

000710E WMQ CICS Bridge MQI call failed

Explanation: An error occurred in an MQI call issued by the bridge.

System action: Depending on the failure, the bridge related task may proceed or issue an abend.

User response: Use CICS trace facilities to identify the cause on the failure and refer to the WebSphere MQ Application Programming manual for an explanation. Take appropriate action. Also refer to the WMQ System Log for associated messages.

000711E WMQ CICS Bridge unable to open bridge queue

Explanation: The bridge queue specified is not known to the queue manager.

System action: WMQ CICS bridge monitor terminated.

User response: Check the bridge queue is defined correctly and that the correct queue is identified at bridge startup.

000712I WMQ CICS Bridge quiescing

Explanation: Monitor quiesce has been initiated. This would normally be because CICS or WMQ is shutting down or because the operator has set the bridge queue GET(DISABLED).

System action: WMQ CICS bridge monitor is stopping.

User response: None.

000713I WMQ CICS Bridge terminated normally

Explanation: Monitor shutdown completed normally.

System action: WMQ CICS bridge monitor shutdown.

User response: None.

000715E WMQ CICS Bridge invalid COMMAREA length in message

Explanation: The COMMAREA length calculated by the bridge is not valid. It probably exceeds the maximum of 32767. This error can also occur if a negative length was calculated.

System action: Message cannot be processed.

User response: If OutputDataLength is set within the MQCIH, check it does not exceed 32759 (allowing 8 bytes for the program name). If it is not set, check the total request message length (also allowing bytes for the program name). The length of any MQCIH must not exceed 32767. Note that the length of the MQCIH is taken from the MQCIH length field.

000716E WMQ CICS Bridge MQCIH required for UOW middle and last messages

Explanation: A bridge task has received a message for a second or subsequent MQGET call within a multipart unit of work. The correlation identifier matches the message identifier of the first message within the unit of work, but the message does not contain an MQCIH.

System action: The unit of work is backed out.

User response: Make sure that all messages within a multipart unit of work contain an MQCIH and rerun the unit of work.

000717E WMQ CICS Bridge UOW first/only received, UOW middle/last expected

Explanation: A bridge task has received a message for a second or subsequent MQGET call within a multipart unit of work. The correlation identifier matches the message identifier of the first message within the unit of work, but the UOWControl field within the MQCIH is invalid. It is set to MQCUOWC_FIRST or MQCUOWC_ONLY when MQCUOWC_MIDDLE, MQCUOWC_LAST, MQCUOWC_COMMIT, or MQCUOWC_BACKOUT is required.

System action: The unit of work is backed out.

User response: Correct the UOWControl field and rerun the unit of work.

000718E WMQ CICS Bridge UOW middle/last received, UOW first/only expected

Explanation: The bridge monitor has received a request message for a new unit of work, the correlation identifier is set to MQCI_NEW_SESSION but the UOWControl field within the MQCIH is set to something other than MQCUOWC_FIRST or MQCUOWC_ONLY.

System action: The unit of work is backed out.

User response: Correct the UOWControl field and rerun the unit of work.

000720E WMQ CICS Bridge Authentication option requires ESM

Explanation: An attempt has been made to start the bridge monitor with AUTH=IDENTIFY or VERIFY_ but security is not active for the CICS system.

System action: WMQ CICS Bridge monitor terminated.

User response: Activate security, or choose a different authentication option.

000721E WMQ CICS Bridge invalid MQCIH

Explanation: A message has been received by the bridge with a MQMD format field of MQFMT_CICS but the data does not begin with a valid MQCIH. Either the StrucId Version, or StrucLength is incorrect.

System action: Bridge message rejected.

User response: Check the version of the header and compare with the level supported by the bridge. Correct the format or the user data as appropriate.

000722E WMQ CICS Bridge invalid message removed from bridge queue

Explanation: This message is issued during monitor initialization. The first message on the queue should be a request to start a unit of work, that is, it should have correlation identifier of MQCI_NEW_SESSION. The monitor removes any messages preceding the first MQCI_NEW_SESSION, copies them to the dead-letter queue and issues this message.

System action: Message is moved to DLQ.

User response: If this is not caused by a failure for a previous request within a unit of work that has already been reported and actioned, correct the request message and rerun the unit of work.

000723E WMQ CICS Bridge task no longer active

Explanation: An unexpected error has occurred in a bridge task causing it to terminate without notifying the monitor. The monitor has detected this and issue this message during recovery processing.

System action: Bridge processing continues

User response: Investigate the cause of the bridge failure by examining any error messages and dumps for the failed task.

000724E WMQ CICS Bridge queue must be defined as local

Explanation: The bridge queue specified is not defined as a local queue.

System action: WMQ CICS Bridge terminated.

User response: Redefine the bridge request queue as a local queue.

000725I WMQ CICS Bridge queue not persistent by default

Explanation: The bridge queue is defined with DEFPSIST(NO). Request messages should be persistent to guarantee that they will be processed.

System action: WMQ CICS Bridge processing continues.

User response: None.

000726I WMQ CICS Bridge queue backout count not hardened

Explanation: The bridge queue is defined with NOHARDENBO.

System action: WMQ CICS Bridge processing continues.

User response: Alter the queue definition to set HARDENBO. The queue should be defined with HARDENBO to ensure that the bridge does not try to process a unit of work a second time following a CICS emergency restart.

000727I WMQ CICS Bridge queue should be FIFO

Explanation: The bridge queue is defined with PRIORITY message delivery sequence. Processing of high priority messages could be delayed if they are added to the queue ahead of the monitor's browse cursor.

System action: WMQ CICS Bridge processing continues.

User response: Alter the queue definition to set MSGDLVSQ(FIFO)

000728E WMQ CICS Bridge queue already open

Explanation: An MQINQ call for the bridge queue found that another process had the queue open for input. This is not allowed when the monitor starts. the queue ahead of the monitor's browse cursor.

System action: WMQ CICS Bridge terminated.

User response: Check that no monitor task is already active for this queue. If no monitor is active check if any bridge tasks that were started by a previous monitor are still active.

000729I WMQ CICS Bridge no dead-letter queue defined to queue manager

Explanation: There is no dead-letter queue defined to the queue manager. The bridge will be terminated if any error occurs that would result in a message being sent to the dead-letter queue.

System action: WMQ CICS Bridge processing continues.

User response: Alter the queue manager to define a dead-letter queue if dead-letter processing is required.

000730I WMQ CICS Bridge unable to open dead-letter queue

Explanation: The dead-letter queue defined to the queue manager could not be opened. The bridge will be terminated if any error occurs that would result in a message being sent to the dead-letter queue

System action: WMQ CICS Bridge processing continues.

User response: Check that the queue manager's dead-letter queue is defined correctly and is available.

000731I WMQ CICS Bridge unable to inquire on dead-letter queue

Explanation: An MQINQ call on the dead-letter queue failed. The bridge will be terminated if any error occurs that would result in a message being sent to the dead-letter queue.

System action: WMQ CICS Bridge processing continues.

User response: Check that the queue manager's dead-letter queue is defined correctly and is available.

000732I WMQ CICS Bridge unable to put message to dead-letter queue

Explanation: An MQPUT to the dead-letter queue failed. If this error occurs in a bridge task, the unit of work is backed out. If this error occurs in the monitor, the monitor will be abnormally terminated.

System action: WMQ CICS Bridge processing continues.

User response: Check that the queue manager's dead-letter queue is defined correctly and is available.

000733I WMQ CICS Bridge dead-letter queue not defined as usage normal

Explanation: The dead-letter queue is not defined correctly. The bridge will be terminated if any error occur that would result in a message being sent to the dead-letter queue.

System action: WMQ CICS Bridge processing continues.

User response: Check that the queue manager's dead-letter queue is defined correctly and is available.

000734I WMQ CICS Bridge dead-letter queue max message length too small

Explanation: The maximum message length allowed for the dead-letter queue is less than the size of the dead-letter header, MQDLH. The bridge will be terminated if any error occurs that would result in a message being sent to the dead-letter queue

System action: WMQ CICS Bridge processing continues.

User response: Check that the queue manager's dead-letter queue is defined correctly.

000735I **WMQ CICS Bridge detected CICS or MQ quiesce**

Explanation: The bridge task received a quiescing return code from an MQOPEN call of the request queue or an MQGET call for the first message within a unit of work.

System action: The request will be processed when CICS, WMQ, or the monitor are restarted.

User response: None.

000736I **WMQ CICS Bridge quiesced before task started**

Explanation: The bridge quiesced before a bridge task could get the first message within a unit of work.

System action: The request will be processed when the monitor is restarted.

User response: None.

000737E **WMQ CICS Bridge detected CICS or MQ quiesce, task backed out**

Explanation: The bridge task received a quiescing return code from an MQGET for a second or subsequent message within a unit of work.

System action: The unit of work is backed out and the bridge task terminated.

User response: Rerun the unit of work.

000738E **WMQ CICS Bridge quiesced, task backed out**

Explanation: The bridge task quiesced while a bridge task was waiting to get a second or subsequent message within a unit of work because the queue was not enabled for getting messages.

System action: The unit of work is backed out and the bridge task terminated.

User response: Rerun the unit of work.

000739E **WMQ CICS Bridge terminated, timeout interval expired**

Explanation: The bridge task did not receive a second or subsequent message for a unit of work within the wait interval specified (or as overridden on the first request for the unit of work) at monitor startup.

System action: The unit of work is backed out and the bridge task terminated.

User response: Increase the WAIT parameter on monitor startup, correct the program that failed to send a subsequent request for a unit of work, or set the UOWControl field correctly for the previous request.

000740E **WMQ CICS Bridge client application requested backout**

Explanation: The bridge task backed out a unit of work on receipt of a MQCUOWC_BACKOUT request.

System action: Unit of work is backed out.

User response: None.

000741E **WMQ CICS Bridge waiting for bridge tasks to complete**

Explanation: This message is issued during monitor quiesce if bridge tasks are found on the monitor's started or active task lists.

System action: Bridge waits for bridge tasks to complete.

User response: None.

000742I **WMQ CICS Bridge message on queue but task not yet started**

Explanation: This message is issued at the end of monitor quiesce. The monitor delayed to allow bridge tasks time to quiesce and is now listing those still outstanding.

System action: Bridge quiesce continues.

User response: None.

000743I **WMQ CICS Bridge task active at quiesce**

Explanation: This message is issued at the end of monitor quiesce. The monitor delayed to allow bridge tasks time to quiesce. The bridge task is probably in a wait in a user program or in a long MQGET wait for a second or subsequent message within a unit of work.

System action: Bridge quiesce continues.

User response: Investigate why the bridge task is still active.

000744E **WMQ CICS Bridge monitor terminated with bridge tasks active**

Explanation: This message is issued at the end of monitor quiesce. The monitor delayed to allow bridge tasks time to quiesce but one or more bridge tasks are still active.

System action: Bridge quiesce complete.

User response: If the bridge tasks are in MQGET waits, consider reducing the WAIT interval on monitor

startup to avoid this situation in future. Note that the monitor cannot be restarted until the bridge tasks terminate.

000745E WMQ CICS Bridge unable to put message to reply queue

Explanation: An MQPUT call to the reply-to queue failed. The response message will be sent to the dead-letter queue.

System action: Reply message sent to dead-letter queue.

User response: Use CICS trace facilities to identify the cause on the failure and refer to the WebSphere MQ Application Programming manual for an explanation. Take appropriate action. Also refer to the WMQ System Log for associated messages.

000746E WMQ CICS Bridge invalid CCSID

Explanation: A request message was received with an invalid value for the CCSID field in the MQMD.

System action: Request message rejected.

User response: Correct the MQMD and reissue the request.

000747E WMQ CICS Bridge invalid encoding

Explanation: A request message was received with an invalid value for the encoding field in the MQMD.

System action: Request message rejected.

User response: Correct the MQMD and reissue the request.

000748E WMQ CICS Bridge request message removed during backout processing

Explanation: The bridge has sent this request message to the dead-letter queue during backout processing.

System action: Request removal backed out.

User response: See the associated messages to determine the cause of the problem.

000749E WMQ CICS Bridge authentication error

Explanation: The monitor is being run with AUTH=VERIFY_UOW or AUTH=VERIFY_ALL. An EXEC CICS SIGNON or EXEC CICS VERIFY PASSWORD command failed.

System action: Request is not processed.

User response: Use CICS trace facilities to identify the cause on the failure and refer to the CICS Application Programming manual for an explanation. Take appropriate action.

000750E WMQ CICS Bridge monitor internal logic error

Explanation: An unexpected condition was detected by the bridge.

System action: Bridge task terminated.

User response: Contact your IBM support center if the problem persists.

000751E WMQ CICS Bridge unable to LINK to program

Explanation: An EXEC CICS LINK command for the user requested program failed.

System action: Program is not parted.

User response: Check that the correct program is requested, and that it is available and correctly defined to CICS.

000752E WMQ CICS Bridge queue cannot be used for reply-to queue

Explanation: The reply-to queue name in a request message is the same as the bridge-request queue name. This is not allowed.

System action: Request is not processed.

User response: Specify a different reply-to queue in the request.

000753E WMQ CICS Bridge message already processed

Explanation: The bridge already attempted to process this request but the request failed and was backed out. This could be because backout processing failed for a bridge task that ended abnormally or because there was a CICS failure while this request was in progress. No attempt is made to process the request a second time.

System action: Request is not processed.

User response: Look at previous error messages for this message on the System Log to determine the cause for the previous failure, and rerun the request.

000754E WMQ CICS Bridge task abend

Explanation: A bridge task terminated abnormally.

System action: Bridge task terminated.

User response: The associated transaction dump can be used to assist problem determination. Correct the problem and rerun the unit of work.

000755E WMQ CICS Bridge queue is not shareable

Explanation: The bridge request queue does not have the SHARE attribute.

System action: Bridge monitor terminated.

User response: Alter the queue definition and restart the monitor.

000756E WMQ CICS Bridge dead-letter queue must be defined as local

Explanation: The bridge request queue does not have the SHARE attribute.

System action: Bridge monitor processing continues.

User response: The dead-letter queue is not defined as a local queue. The bridge will be terminated if any error occurs that would result in a message being sent to the dead-letter queue.

000757E WMQ CICS Bridge unable to open reply-to queue

Explanation: The reply to queue specified is not known to the queue manager.

System action: Request is not processed.

User response: Check that the reply-to queue is correct, and it is available and correctly defined to the queue manager.

000758E WMQ CICS Bridge unable to START bridge task

Explanation: The monitor is being run with the IDENTIFY or VERIFY authorization option. An EXEC CICS START command for the bridge task failed with NOTAUTH or USERIDERR because the user ID is not authorized to start bridge transactions or has been revoked.

System action: Request is not processed.

User response: Correct the security definitions if this userid should be authorized to run requests using the bridge.

000759E WMQ CICS Bridge transaction not defined to CICS

Explanation: A request has been received to run the transaction listed but it is not defined to this CICS system.

System action: Request is not processed.

User response: Correct the request or define the transaction.

000764I WMQ CICS Bridge invalid userid

Explanation: A user ID is required in all request messages when AUTH=VERIFY_ALL is being used; this must be the same for all requests within a unit of work. This message is issued because the bridge task detected a missing or changed user ID.

System action: UOW backed out.

User response: Correct the user ID and rerun the unit of work.

000799E WMQ CICS Bridge unexpected error

Explanation: An unexpected error has occurred. This message may be generated if an error occurs before the bridge error processing logic is initialized.

System action: Bridge task is terminated.

User response: Use CICS trace facilities to identify the cause on the failure and refer to the CICS Application Programming manual for an explanation. Check the MQ System Log for associated messages.

001000I Function completed

Explanation: The requested function has been completed

System action: Function is completed.

User response: None.

001090E Function not completed

Explanation: The requested function was terminated because of error. The function was not completed.

System action: Function is terminated with error.

User response: Review the associated message prior to this one.

005000I Channel connected

Explanation: Channel connection is successful.

System action: platform negotiation will begin.

User response: None.

002000I System monitor has detected an unowned connection

Explanation: The queue manager housekeeping task has identified a MQI connection for a task that is no longer active.

System action: The connection is ended, and associated resources are freed.

User response: None.

002010I System monitor has detected an inactive object handle

Explanation: The queue manager housekeeping task has identified an object handle (HOBJ) for an MQI connection that no longer exists.

System action: The object handle is closed, and associated resources are freed.

User response: None.

005001I Channel negotiations accepted

Explanation: Channel has completed negotiation with the other platform.

System action: Message queue will be opened.

User response: None.

005002I Channel queue opened

Explanation: Channel queue has been opened successfully.

System action: Message transfer will begin.

User response: None.

005003I LU 6.2 connection established

Explanation: LU 6.2 connection established.

System action: LU 6.2 conversation will begin.

User response: None.

005004I Channel receiver allocated

Explanation: (Reserved)

005005I Channel queue empty

Explanation: Sender finds the queue is empty

System action: Transmission queue will be closed and the Channel will be disconnected and shutdown after number of get retries exhausted.

User response: None.

005006I Channel queue closed

Explanation: Channel has successfully closed queue.

System action: Channel will be disconnected.

User response: None.

005007I Channel disconnected

Explanation: Channel has been disconnected from the other platform.

System action: Channel will be shutdown.

User response: None.

005008I Channel shutdown

Explanation: Channel has been completely shutdown.

System action: Channel is marked INACTIVE.

User response: None.

005009I Channel shutdown request sent

Explanation: (Reserved)

006003I TCP/IP channel connected

Explanation: TCP/IP connection established.

System action: TCP/IP conversation will begin.

User response: None.

006007I TCP/IP session started

Explanation: A communication session was started by the MQI Receiver MCA.

System action: TCP/IP conversation will begin.

User response: None.

006010E TCP/IP connection broken

Explanation: A TCP/IP connection terminated prematurely, possibly due to incomplete data from remote partner, or the premature closure of an active TCP/IP socket.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message.
2. Check host TCP/IP services are active.
3. Check TCP/IP error log on remote system.
4. Correct problem and restart channel.

006013E TCP/IP storage allocation error For Program MQPTCPSV -

Explanation: The MQ/Server program was unable to allocate memory.

System action: Fatal error - Communication is terminated.

User response: This error occurs when there is insufficient memory resources available. Check that other processes are not erroneously allocating memory, or rerun the client/server conversation when the z/VSE host is less busy.

006015E TCP/IP connection error For Program MQPSEND -

Explanation: As a Sender upon completion of a CONNECT request, received an invalid return code.

System action: Fatal error - Communication is not established.

User response:

1. Review System Log or error TD Queue for messages prior to this message.
2. Check that the sender channel port number matches the listener port of the receiver.
3. Check that the TCP Partner is a valid host or IP address n.n.n.n of the receiving system.
4. Correct problem and restart communication.

006016E TCP/IP send error

Explanation: As a Sender or Server upon completion of a data send request received an invalid return code.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message.
2. Produce an auxiliary trace to determine the return code returned from the send.
3. Correct problem and restart communication.

006017E TCP/IP receive/respond error

Explanation: An attempt to receive TCP/IP data failed or the response from a remote system was other than as expected.

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message.
2. Check the TCP/IP error log on the remote system to see if the channel was closed prematurely.
3. Correct problem and restart communication.

006020E TCP/IP socket error

Explanation: An error occurred when opening or setting the attributes of a TCP/IP socket.

System action: Fatal error - communication cannot start.

User response:

1. Check that TCP/IP is installed and running.
2. Check that the TCP/IP phase is cataloged in a library that is concatenated ahead of SCEEBASE in the LIBDEF of the CICS startup deck.

006021E TCP/IP transport error

Explanation: A TCP/IP conversation ended due to a transport protocol error, or the use of a conversation was attempted before it was established.

System action: Fatal error - communication is terminated.

User response:

1. Check that TCP/IP is running.
2. Attempt to restart the conversation.

006022C TCP/IP listener bind error

Explanation: The MQ Listener program attempted to bind a port number to a TCP/IP socket and was unsuccessful.

System action: Fatal error - Listener cannot start.

User response:

1. Check that the Listener is not already running.
2. Check that another application is not using the port number configured for the Listener.

006023C TCP/IP listener accept error

Explanation: The MQ Listener program attempted to accept a remote conversation and failed.

System action: Error - Listener is terminated.

User response:

1. This is not an error if the remote program terminated before the conversation was accepted.
2. Check that TCP/IP is running.
3. Restart the Listener.

006024C TCP/IP listener error

Explanation: The MQ Listener program terminated due to an unexpected error.

System action: Fatal error - Listener is terminated.

User response:

1. Check that TCP/IP is running.
2. Examine the system log for associated error messages.
3. Restart the Listener.

006025I TCP/IP listener stopped

Explanation: The MQ Listener program terminated normally or due to an error.

System action: Listener program terminates.

User response:

1. Check the previous log entries for error messages. If no previous error messages then the Listener terminated normally.
2. Restart the Listener when appropriate.

006026E TCP/IP invalid channel type

Explanation: The channel type value in a channel definition is invalid.

System action: Error - Channel cannot be started.

User response:

1. Check channel definition documentation for valid channel type values. Then update the channel definition appropriately.

006027E TCP/IP channel negotiation failed

Explanation: The conversation initial negotiation failed.

System action: Error - channel cannot be started.

User response:

1. Check that the local and remote channel definitions are compatible.

006028E TCP/IP channel protocol error

Explanation: The protocol type value in a channel definition is invalid.

System action: Error - Channel cannot be started.

User response:

1. Check channel definition documentation for valid protocol type values. Then update the channel definition appropriately.

006029E TCP/IP connect failed

Explanation: An attempt to establish a TCP/IP connection failed.

System action: Fatal error - conversation cannot be started.

User response:

1. Check channel definition to ensure the hostname or IP address and port number are valid for the intended remote host.
2. Check that the remote system's listener process if active.

006030E TCP/IP unknown remote channel name

Explanation: The channel name used in a remote conversation does not exist on the remote host.

System action: Error - communication is terminated.

User response:

1. Create the channel definition on the remote host.

006031E TCP/IP remote queue manager not available

Explanation: The queue manager identified in for a remote host is currently unavailable.

System action: Error - communication is terminated.

User response:

1. Start the remote queue manager on the remote host. :
2. Retry the channel.

006032W TCP/IP channel stopped by operator/application

Explanation: The channel being used in a remote conversation has been disabled by an operator or application.

System action: Error - communication is terminated.

User response:

1. Restart the channel when appropriate.

006033E TCP/IP channel not active

Explanation: An attempt was made to use a channel that is not currently started.

System action: Error - communication is terminated.

User response:

1. Start the channel when appropriate.

006034I TCP/IP channel stopped

Explanation: The channel stopped normally or due to an error.

System action: Channel activity is terminated.

User response:

1. Check previous log messages for errors. If there are no errors, then the channel stopped normally.
2. Restart the channel when appropriate.

006035C TCP/IP syncpoint failed

Explanation: An attempt to perform a CICS SYNCPOINT failed during a TCP/IP conversation.

System action: Transaction changes are rolled back to the beginning of the current unit of work.

User response:

1. Check the system log for previous error messages. :
2. Check CICS logs for possible errors or insufficient resources.

006036W TCP/IP message put to DLQ

Explanation: A message could not be delivered and was instead written to the system dead letter queue.

System action: Message written to system dead letter queue.

User response:

1. Examine the dead letter queue for the undelivered message.

006037E TCP/IP invalid remote channel type

Explanation: The channel type value in a channel definition on a remote host is invalid.

System action: Error - Channel cannot be started.

User response:

1. Check channel definition documentation for valid channel type values. Then update the channel definition on the remote host appropriately.

006038W TCP/IP transmission queue get inhibited

Explanation: The transmission queue identified by a channel definition has GET INHIBIT enabled.

System action: Error - messages cannot be retrieved and sent.

User response:

1. Disable GET INHIBIT on the transmission queue.

006039E TCP/IP remote channel unavailable

Explanation: The channel identified for a remote conversation is unavailable.

System action: Error - channel cannot be started.

User response:

1. Start the channel on the remote host.

006040E TCP/IP channel abnormally ended

Explanation: The channel involved in a current conversation was terminated with an error, possible due a remote partner ending the conversation prematurely.

System action: Fatal error - communication is terminated.

User response:

1. Check the system log for previous error messages.

006041I TCP/IP listener started

Explanation: The MQ Listener program started.

System action: MQ Listener ready to accept remote connections.

User response: None.

006042I TCP/IP server started

Explanation: An MQ Server program instance started in response to a remote client connection.

System action: MQ Server ready to process client requests.

User response: None.

006043I TCP/IP server stopped

Explanation: An MQ Server program instance stopped normally or due to an error.

System action: Client conversation terminated.

User response:

1. Check previous system log messages to determine whether this is a normal termination or due to an error.

006044E TCP/IP bad server commarea

Explanation: An MQ Server program instance was started with an invalid commarea.

System action: Fatal error - MQ Server program terminated.

User response:

1. Check the MQ Server program is not being started by an application other than the MQ Receiver MCA.

006045W TCP/IP server error

Explanation: An error occurred during an MQ client conversation.

System action: Client conversation completed with errors.

User response:

1. Check the system log for previous error messages and respond to these.

006046C WMQ Server environment error

Explanation: An error occurred during an MQ server activation. The MQ Server could not load relevant program modules.

System action: Client conversation is terminated.

User response: Check the queue manager has been

installed correctly, and that MQ modules are correctly defined to CICS.

006047W Data conversion code page error

Explanation: Data conversion from source code page to target code page is not supported.

System action: The message data is not converted.

User response: Check the requested code pages are compatible and change if necessary.

006050W Data conversion source code page error.

Explanation: The value in the source code page is an unknown value.

System action: The message data is not converted.

User response: Check the source code page is valid.

006053W Data conversion target code page error

Explanation: The value in the target code page is an unknown value.

System action: The message data is not converted.

User response: Check the target code page is valid.

006054W Default data conversion code page error

Explanation: Default data conversion between the specified code pages is not supported.

System action: The message data is not converted.

User response: Check the default code pages specified in the code page definition panel are compatible.

006055W Data conversion IMS string error

Explanation: The passed IMS string is the wrong length for data conversion.

System action: The IMS message data is not converted.

User response: Change the IMS string length.

006057W Data conversion PCF header bad length

Explanation: The passed PCF header is the wrong length.

System action: The PCF message data is not converted.

User response: Change the PCF header string length. The buffer is too small for a complete PCF header.

006058E Data conversion bad ICONV return code for LE conversion

Explanation: Data conversion failed.

System action: The message data is not converted.

User response:

1. Ensure LE/VSE code page is available.
2. Review LE ICONV family services and ensure product features are installed and functional.

006059I Data conversion target code page set to source code page

Explanation: The target code was set to null.

System action: Data conversion continues.

User response: Ensure the target code page is correctly set.

006060W Data conversion is not supported

Explanation: Data conversion is not supported between the specified code pages.

System action: Conversion fails.

User response: Change the code pages to a pair which are supported for LE data conversion.

006063C Data conversion error finding MQ internal control block

Explanation: WMQ for z/VSE is not initialized correctly.

System action: Data conversion ends.

User response:

1. Stop and restart WebSphere MQ. Check WebSphere MQ and CICS logs for possible errors such as SOS.
2. If product maintenance has been recently applied, review cover letter for installation steps that may have been missed.

006100E TCP/IP SSL initialization failed

Explanation: Attempt to establish a secure socket connection using SSL failed.

System action: Channel will not initialize.

User response:

1. Check that remote system requesting the secure socket connection uses standard SSL protocol and valid PKI certificate.
2. For a sending system, check that local SSL services are installed and configured correctly.

006101E TCP/IP SSL cipher specification error

Explanation: Secure socket connection abandoned due to channel cipher specification mismatch.

System action: Channel is terminated.

User response:

1. Check that the remote system requesting the secure socket connection identifies a cipher specification that matches the cipher specification of the local receiver channel.
2. Check that local SSL services support the channel cipher specification.

006102E TCP/IP SSL peer attribute error

Explanation: Secure socket connection abandoned due to channel peer attribute mismatch with partner certificate details.

System action: Channel is terminated.

User response:

1. Check that the SSLPEER specification of the local channel identifies the certificate details of the remote system communications partner. (Remember that values are case sensitive).
2. Check that the remote client or sender MCA is an authorized SSL partner.

006103E TCP/IP SSL client authentication error

Explanation: Secure socket connection abandoned due to remote system failing to provide a valid certificate during channel initialization.

System action: Channel is terminated.

User response:

1. If the receiver channel specifies SSLCAUTH is REQUIRED, then the remote client or sender MCA must provide a valid certificate during channel/SSL initial negotiation.
2. If the receiver channel does not require a client certificate, then the receiver channel can set the SSLCAUTH to optional.
3. If the receiver channel requires a peer name match, then the remote system must provide a certificate during channel initialization.

006999E TCP/IP unexpected error

Explanation: An unexpected error has occurred.

System action: Unknown.

User response:

1. Check the system log for previous error messages.
2. Check the QCODE in this error message. This should be a numeric code that will be meaningful to MQ system support.

007000I PCF command server started

Explanation: The PCF command server has been started.

System action: PCF command server ready to process PCF commands from the system command queue.

User response: None.

007001W PCF command server not started

Explanation: The PCF command server could not be started.

System action: PCF command server instance cannot start.

User response: Check the following:

1. Queue manager is active.
2. Command server not already active.
3. System log for further error messages.

007002W PCF command server not started

Explanation: The PCF command server could not be stopped because it is not running.

System action: None.

User response: None.

007003I PCF command server terminated by request

Explanation: The PCF command server has terminated due to an operational request.

System action: The PCF command server is nolonger available to processes PCF commands from the command queue.

User response: Restart the command server when necessary.

007004I PCF command server termination requested

Explanation: An operational request to terminate the PCF command server has been received.

System action: The PCF command server will terminate, and can be restarted when necessary.

User response: None.

007005W PCF command server already running

Explanation: An request to start the PCF command server was received when the server is already running.

System action: Only one instance of the command server can be running at any time. The initial instance

will continue to process command messages until terminated.

User response: None.

007006C Insufficient storage for PCF command server

Explanation: An attempt to allocate CICS storage for the PCF command server during initialization failed.

System action: The PCF command server cannot be started.

User response: The command server attempts to allocate approximately 2k of CICS storage during initialization. Ensure the relevant CICS region has sufficient storage resources.

007007C PCF command server cannot process command queue

Explanation: An attempt by the PCF command server to issue an MQGET from the system command queue failed.

System action: The PCF command server is terminated.

User response: Check the status of the system command queue to ensure it is enabled and available for processing.

007008C PCF command server cannot connect to queue manager

Explanation: An attempt by the PCF command server to issue an MQCONN to establish an MQI session with the local queue manager failed.

System action: The PCF command server is terminated.

User response: Check the status of the queue manager to ensure it is active and available for MQI connectivity.

007009C PCF command server cannot open command queue

Explanation: An attempt by the PCF command server to issue an MQOPEN of the system command queue failed.

System action: The PCF command server is terminated.

User response: Check the status of the system command queue to ensure it is enabled and available for processing.

007010W PCF command server stopped due to system quiescing

Explanation: The PCF command server has detected that the queue manager is quiescing or stopping.

System action: The PCF command server is terminated.

User response: No action is required if the queue manager has been stopped intentionally. If not, the cause of the queue manager stopping should be investigated, and the PCF command server restarted if a manual start is required.

007011E PCF command server stop request failed

Explanation: An attempt to stop the PCF command server failed.

System action: The PCF command server will not terminate.

User response: Check that the system command queue is configured and able to accept command messages. The queue should not be PUT inhibited or full. Also, if MQ security is enabled, check that the stop request was issued by a userid with sufficient authority.

007012C PCF command server cannot start command processor

Explanation: The PCF command server received a PCF command on the system command queue but was unable to start the command processor to process the command.

System action: The PCF command server is terminated.

User response: Check that the command processor is correctly defined to the CICS system. Also check that the CICS system has sufficient resources to start new transactions.

007013W PCF command server insufficient authority to start processor

Explanation: The PCF command server received a PCF command on the system command queue but was unable to start the command processor to process the command due to an authorization failure.

System action: The PCF command server rejects the command.

User response: Check that the userid running the command server has sufficient authority to start the command processor transaction as any user with authority to put messages to the system command queue. If security is enabled, the command server user must be a surrogate user for users that can put messages to the command queue.

007014W PCF command server could not put reply to DLQ

Explanation: The PCF command server attempted to put a PCF reply message to the system dead letter queue and the attempt failed.

System action: The PCF reply message is lost.

User response: Check that the system dead letter queue is properly configured for the queue manager. Also check that the dead letter queue is not full or inhibited.

007015W PCF command server could not be auto-started

Explanation: An attempt to automatically start the PCF command server failed during system initialization.

System action: The PCF command server is not auto-started.

User response: Check that the command server is not already running and that the supplied MQ transactions and programs have been installed correctly.

007016W Batch interface could not be auto-started

Explanation: An attempt to automatically start the batch interface failed during system initialization.

System action: The batch interface is not auto-started.

User response: Check that the batch interface is not already running and that the supplied MQ transactions and programs have been installed correctly.

007017I PCF command server stopped

Explanation: The PCF command server has been stopped.

System action: PCF command server is no longer available to process PCF commands from the command queue.

User response: The PCF command server may have stopped due to an error or an operational request. In case of an error, check the system Log for previous error messages.

007020E PCF command processor could not connect to queue manager

Explanation: An attempt to connect to the local queue manager failed.

System action: The PCF command processor attempts to connect to the local queue manager when it is ready to send a PCF response message. The response message is lost, and the result of the original PCF command must be determined manually.

User response: Check that the queue manager is active and is configured to accept a suitable number of concurrent connection requests.

007021W PCF command processor could not open reply queue

Explanation: An attempt to open a locally defined queue for a PCF response message failed.

System action: If the PCF command processor is configured to put undeliverable response message to the system dead letter queue, then an attempt to do so is made. Otherwise, the response message is lost and the result of the original PCF command must be checked manually.

User response: Check that the ReplyToQ and ReplyToQMgr fields of the MQMD of the originating PCF command message identify a valid and available queue.

007022W PCF command processor could not send response message

Explanation: An attempt to put a PCF response message to the relevant ReplyToQ failed.

System action: If the PCF command processor is configured to put undeliverable response message to the system dead letter queue, then an attempt to do so is made. Otherwise, the response message is lost and the result of the original PCF command must be checked manually.

User response: Check that the relevant queue is available for the receipt of messages.

007023C PCF command processor could not allocate storage

Explanation: An attempt by the PCF command processor to allocate CICS storage failed.

System action: The PCF command processor cannot allocate the necessary storage to generate a PCF response message, and so the response message is lost. The result of the original PCF command must be checked manually.

User response: Check that the relevant CICS partition has sufficient above the line storage to satisfy GETMAIN requests.

007024E PCF command processor could not divert to DLQ

Explanation: An attempt by the PCF command processor to put an undeliverable PCF response message to the system dead letter queue failed.

System action: The PCF response message is lost and the result of the original PCF command must be checked manually.

User response: Check that the system dead letter queue is available and ready to receive messages.

007100I MQ SOAP listener started

Explanation: The MQ SOAP listener started on request queue.

System action: MQ SOAP listener ready to process MQ SOAP requests.

User response: None.

007101W MQ SOAP listener not started

Explanation: The MQ SOAP listener failed to start.

System action: MQ SOAP listener not started.

User response: Check for previous error messages.

007102E MQ SOAP listener not started due to invalid parameter

Explanation: The Q= or SOAPPORT= parameter is not valid.

System action: MQ SOAP listener is not started.

User response: Check that there is only one Q= and specified request queue name is valid. The SOAPPORT= specifies the CICS SOAP port number and be in range 1 through 65535.

007104I MQ SOAP listener termination requested

Explanation: An operational request to terminate the MQ SOAP listener has been received.

System action: The MQ SOAP listener will terminate, and can be restarted when necessary.

User response: None.

007106C Insufficient storage for MQ SOAP listener

Explanation: An attempt to allocate CICS storage for the MQ SOAP listener during initialization failed.

System action: The MQ SOAP listener cannot be started.

User response: The MQ SOAP listener attempts to allocate approximately 2k of CICS storage during initialization. Ensure the relevant CICS region has sufficient storage resources.

007107C MQ SOAP listener cannot process request queue

Explanation: An attempt by the MQ SOAP listener to issue an MQGET from the request queue failed.

System action: The MQ SOAP listener is terminated.

User response: Check the status of the MQ SOAP request queue to ensure it is enabled and available for processing.

007108C MQ SOAP listener cannot connect to queue manager

Explanation: An attempt by the MQ SOAP listener to issue an MQCONN to establish an MQI session with the local queue manager failed.

System action: The MQ SOAP listener is terminated.

User response: Check the status of the queue manager to ensure it is active and available for MQI connectivity.

007109C MQ SOAP listener cannot open request queue

Explanation: An attempt by the MQ SOAP listener to issue an MQOPEN of the request queue failed.

System action: The MQ SOAP listener is terminated.

User response: Check the status of the request queue to ensure it is enabled and available for processing.

007110W MQ SOAP listener stopped due to system quiescing

Explanation: The MQ SOAP listener has detected that the queue manager is quiescing or stopping.

System action: The MQ SOAP listener is terminated.

User response: No action is required if the queue manager has been stopped intentionally. If not, the cause of the queue manager stopping should be investigated, and the MQ SOAP listener restarted if a manual start is required.

007111E MQ SOAP listener stop request failed

Explanation: An attempt to stop the MQ SOAP listener failed.

System action: The MQ SOAP listener will not terminate.

User response: Check that the request queue is configured and able to accept messages. The queue should not be PUT inhibited or full. Also, if WMQ security is enabled, check that the stop request was issued by a userid with sufficient authority.

007112C MQ SOAP listener cannot start SOAP request processor

Explanation: The MQ SOAP listener received a MQ SOAP request on the request queue but was unable to start the SOAP request processor to process the request.

System action: The MQ SOAP request is placed on dead letter queue.

User response: Check that the MQ SOAP processor is correctly defined to the CICS system. Also check that the CICS system has sufficient resources to start new transactions.

007113W MQ SOAP listener has insufficient authority to start processor

Explanation: The MQ SOAP listener received a MQ SOAP request on the request queue but was unable to start the MQ SOAP processor to process the request due to an authorization failure.

System action: The MQ SOAP listener rejects the request.

User response: Check that the userid running the MQ SOAP listener has sufficient authority to start the MQ SOAP processor transaction as any user with authority to put messages to the MQ SOAP request queue. If security is enabled, the MQ SOAP listener user must be a surrogate user for users that can put messages to the request queue.

007114W MQ SOAP listener could not put request to DLQ

Explanation: The MQ SOAP listener attempted to put the request message to the system dead letter queue and the attempt failed.

System action: The MQ SOAP request message is lost.

User response: Check that the system dead letter queue is properly configured for the queue manager. Also check that the dead letter queue is not full or inhibited.

007117I MQ SOAP listener stopped

Explanation: The MQ SOAP listener has been stopped.

System action: MQ SOAP requests put on the specified request queue will not be processed.

User response: The MQ SOAP listener may have stopped due to an error or an operational request. In case of an error, check the system Log for previous error messages.

007220E MQ SOAP handler cannot connect to queue manager

Explanation: An attempt by the MQ SOAP handler to issue an MQCONN to establish an MQI session with the local queue manager failed.

System action: The MQ SOAP handler is terminated.

User response: Check the status of the queue manager to ensure it is active and available for MQI connectivity.

007221W MQ SOAP handler could not open reply queue

Explanation: An attempt to open a locally defined queue for a MQ SOAP response message failed.

System action: The MQ SOAP handler terminates task.

User response: Check that the ReplyToQ and ReplyToQMgr fields of the MQMD of the originating MQ SOAP request message identify a valid and available queue.

007222W MQ SOAP handler could not send response message

Explanation: An attempt to put a MQ SOAP response message relevant ReplyToQ failed.

System action: The MQ SOAP handler terminates task.

User response: Check that the relevant queue is available for the receipt of messages.

007223C MQ SOAP handler could not allocate storage

Explanation: An attempt by the MQ SOAP handler to allocate CICS storage failed.

System action: The MQ SOAP handler task terminates

User response: Check that the relevant CICS partition has sufficient above the line storage to satisfy GETMAIN requests.

010000W System started with errors

Explanation: System being initialized but some Queue / Channel definitions had errors.

System action: Erroneous Queues / Channels are marked as DISABLED.

User response:

1. Review System Log for prior error messages to identify problem definition.
2. Correct definitions.
3. Shutdown and then reinitialize system.

010001W System started with file errors

Explanation: System being initialized but some Queues files had errors.

System action: Erroneous Queues are marked DISABLED.

User response:

1. Review System Log for prior error messages to identify problem definition.
2. Correct definitions.

3. Shutdown and then reinitialize system.

010002W System started with channel errors

Explanation: System being initialized but some channel definitions had errors.

System action: Erroneous channels are marked DISABLED.

User response:

1. Review System Log for prior error messages to identify problem definition.
2. Correct definitions.
3. Shutdown and then reinitialize system.

010003W System started but system changed

Explanation: System being initialized but definitions have been added / deleted while initialization was being performed.

System action: If definitions were added then some definitions may not be used.

User response: Do not perform configuration changes while system is being initialized. Shutdown and then reinitialize system.

010005I Reorganization process has disabled queue <qname>

Explanation: The specified queue name has been disabled for a scheduled reorganization of the VSAM file.

System action: None.

User response: None required.

010007I Reorganization process has enabled queue <qname>

Explanation: The specified queue name has been enabled after the VSAM file has been reorganized.

System action: None.

User response: None required.

010010I Channel auto-defined

Explanation: A channel which did not previously exist has been auto-defined.

System action: Channel auto-defined successfully.

User response: None.

010011E Channel auto-definition exit unavailable

Explanation: The queue manager's channel auto-definition feature is enabled, but the exit program is unavailable.

System action: Channel auto-definition is disabled.

User response: Define and enable the exit program to your CICS system.

010012E Channel auto-definition model channel unavailable

Explanation: The queue manager's channel auto-definition feature is enabled, but the relevant model channel is not available.

System action: The channel instance is not auto-defined.

User response: Define and enable the relevant model channel.

010013E Channel auto-definition failed

Explanation: The queue manager's channel auto-definition feature is enabled, and an attempt to auto-define a channel failed.

System action: The channel instance is not auto-defined.

User response: Check VSE console and system log for associated error messages.

010014W Channel auto-definition suppressed by exit

Explanation: The queue manager's channel auto-definition exit has returned a failure code.

System action: The channel instance is not auto-defined.

User response: Check exit rules for auto-definition.

010015E Channel auto-definition exit returned invalid values

Explanation: The queue manager's channel auto-definition exit has returned values that are not valid. The exit must not change the channel name, channel type, or the MQCD version.

System action: The channel instance is not auto-defined.

User response: Check exit program conforms to exit rules.

010018I WMQ API exits enabled

Explanation: The queue manager has detected that WMQ API exits are configured and will invoke these exits during normal MQI operation.

System action: WMQ API exits will be invoked during MQI processing.

User response: None.

010020C Global lock error detected by reorganization

Explanation: The global lock table is corrupt for the queue currently being reorganized.

System action: Reorganization is terminated.

User response: Stop and restart WMQ for z/VSE.

010021W Reorganization target queue <qname> is busy

Explanation: The queue is already in use. Automatic Reorganization will stop and retry later.

System action: Reorganization is terminated, but will be retried, if appropriate.

User response: You may wish to reschedule reorganization if all the retries have failed.

010022E Redefine of reorganization file <fname> failed

Explanation: The IDCAMS delete and redefine of the reorganization dataset failed.

System action: Reorganization is terminated.

User response: See CICS console for any error messages from IDCAMS.

010023W Reorganization deleted no records for queue <qname>

Explanation: No records have been deleted from the queue file during automatic reorganization.

System action: Reorganization continues and the queue file will be reallocated.

User response: None.

010024I Reorganization successful for queue <qname>

Explanation: The queue has been reorganized; logically deleted messages have been removed, the underlying VSAM file has been deleted and redefined, and existing messages have been restored.

System action: Queue available for processing.

User response: None.

010025E File <fname> has multiple queues, so reorganization failed for queue <qname>

Explanation: During reorganization it has been detected that the associated VSAM file contains records for more than one queue. Automatic reorganization is only possible for files that contain records for a single queue.

System action: Reorganization terminates.

User response: Please ensure only one queue is defined to this queue file. Alternatively you can reorganize this type of file using the offline batch job MQPREORG.

010026C Rename of file <fname> failed during reorganization

Explanation: The IDCAMS rename for the reorganization file to the queue file failed.

System action: Reorganization finishes.

User response:

1. See CICS console for any error messages from IDCAMS.
2. Manually rename the reorganization file to the queue file.
3. Open the queue file in CICS.
4. Restart the Queue via MQMT.

010027W Automatic reorganization already running

Explanation: Reorganization already running. Only one reorganization can run at one time.

System action: Reorganization is terminated.

User response: Change the scheduled time for reorganization startup if possible.

010028E Reorganization file <fname> not disabled

Explanation: The reorganization file is automatically disabled before it is reallocated, however the attempt to disable the file failed.

System action: Reorganization is terminated.

User response:

1. See CICS console for any error messages.
2. Use CICS to manually disable the file before rescheduling the reorganization.

010029E Reorganization file <fname> not closed

Explanation: The reorganization file is automatically closed before it is reallocated, however the attempt to close the file failed.

System action: Reorganization is terminated.

User response:

1. See CICS console for any error messages
2. Use CICS to manually close the reorganization file before rescheduling the reorganization.

010030E Reorganization file <fname> is not defined to CICS

Explanation: The reorganization file must be defined to CICS to automatically reorganize a queue.

System action: Reorganization is terminated.

User response: Define the reorganization file to CICS as file MQFREOR.

010031E Reorganization cannot read queue file <fname>

Explanation: The queue file must be readable to enable automatic reorganization to take place.

System action: Reorganization is terminated.

User response: Ensure the queue file's CICS definition is set to readable.

010032E Reorganization queue file <fname> not defined

Explanation: The queue file must be defined to CICS to automatically reorganize a queue.

System action: Reorganization is terminated.

User response: Define the queue file to CICS.

010033W Reorganization cannot open queue file <fname>

Explanation: An attempt to open the queue file has failed.

System action: Reorganization continues. See next message.

User response:

1. See CICS console for any error messages.
2. Attempt to open the queue file manually in CICS.

010034E Reorganization file <fname> not open or updateable

Explanation: An attempt to open the reorganization file has failed. The reorganization file must be opened and updateable to enable automatic reorganization to take place.

System action: Reorganization is terminated.

User response:

1. See CICS console for any error messages. and resolve the error opening the file.
2. Ensure the reorganization file's CICS definition is set to updateable and can be opened in CICS.

010035W Reorganization cannot close queue file <fname>

Explanation: An attempt to close the queue file has failed.

System action: Reorganization ends, and the queue file is started without being reorganized.

User response:

1. See CICS console for any error messages and resolve the error closing the file.

010036E Reorganization file <fname> not closed

Explanation: An attempt to close the reorganization file has failed.

System action: Reorganization ends.

User response:

1. See CICS console for any error messages. and resolve the error closing the file.
2. Manually close the file in CICS.
3. Manually rename the IDCAMS reorganization file to the queue file.
4. Open the Queue File in CICS.
5. Restart the queue via MQMT.

010037E Reorganization file <fname> not allocated

Explanation: An attempt to find name of the reorganization file has failed.

System action: Reorganization ends.

User response: Manually reallocate the reorganization file and reschedule the reorganization.

010038E Reorganization failed to delete queue file <fname>

Explanation: An attempt to delete the queue file has failed during reorganization.

System action: Reorganization ends, and the queue file is started without being reorganized.

User response: See CICS console for any error messages from IDCAMS and resolve the error before rescheduling the automatic reorganization.

010039W Deletion of queue failed because queue is not empty

Explanation: An attempt to delete a queue has failed.

System action: Queue is not deleted.

User response: Process queue messages before attempting to delete the queue.

010040C Reorganization file <fname> contains records for queue <qname>

Explanation: The requested reorganization is rejected as the MQFREOR file contains messages for a queue whose file does not exist. This can occur when previous reorganization failed when trying to rename the reorganization file to the queue file.

System action: Reorganization request is not performed, and no further automatic reorganization can take place until this problem is resolved.

User response: Manually rename the reorganization file to the queue file.

010042E MQFREOR file is in different VSAM catalog to <fname> file

Explanation: A reorganization request for a queue is rejected as the file containing the queue is in a different VSAM catalog to the MQFREOR file.

System action: Reorganization request is not performed, and no further automatic reorganization can take place until this problem is resolved.

User response: The queue file to be reorganized and the MQFREOR file have to be in the same VSAM catalog. DELETE/DEFINE the MQFREOR file to be in the same VSAM catalog as the queue file and change JCL or CSD to point to the correct VSAM catalog.

All queue files that are to use automatic reorganization have to be in the same VSAM catalog.

010050E No storage for Accounting and Statistics message generation

Explanation: There is insufficient storage available to allocate the Accounting and Statistics message

System action: The Accounting and Statistics message generation service is disabled.

User response: Check the storage resources available to your CICS region and increase as appropriate.

010051E Cannot start statistics message generation module

Explanation: An attempt to LINK the statistics message generation module (MQPSTATS) has failed.

System action: The statistics message generation service is disabled.

User response: Check EIB return codes that accompany this message to determine the cause of the failure and correct as appropriate.

010052E Accounting Message TDQ not defined or unavailable

Explanation: The transient data queue used for accounting message generation has not been defined, or is unavailable to the WMQ startup user.

System action: Accounting message generation is disabled.

User response: Check that the Accounting Message TDQ has been defined and enabled, and is accessible by the WMQ startup user.

010053E Error delivering Accounting message

Explanation: An attempt to generate an Accouninting message failed.

System action: Accounting message generation is disabled.

User response: Check that the Accounting Message queue is defined and available.

100000W System stopped while in use

Explanation: The queue manager was stopped while applications and/or channels were active.

System action: System stopped, and active applications and channels are terminated.

User response: All applications and channels should be terminated before the system is shutdown.

100010I System already active

Explanation: An attempt to initialize the queue manager failed because the system is already active.

System action: System initialization not performed.

User response:

1. Ignore this message, as an initialization attempt was made in error.
2. Stop and restart the queue manager.

100011W System started with no queues

Explanation: The queue manager has been started, but it has no queue definitions.

System action: System initialized but inoperative.

User response: The queue manager can provide no effective services without any queue definitions. Define queues when possible.

100012W System started with too many queues.

Explanation: The queue manager has been started but has more queue definitions than it can process.

System action: System initialized with some queue definitions.

User response: Delete queue definitions as appropriate.

100013W System started with too many channels

Explanation: The queue manager has been started but has more channel definitions than it can process.

System action: System initialized with some channels.

User response: Delete channel definitions as appropriate.

100090W System started with no queue manager definition

Explanation: The system has been started, but the queue manager definition cannot be found.

System action: System initialization is terminated.

User response: Check that the correct MQ configuration file has been defined to the CICS region. If the queue manager record has been intentionally deleted from the configuration file, a new definition can be created using the MQMT transaction.

100092E Message expiry subsystem failure

Explanation: The message expiry subsystem failed to logically delete an expired message.

System action: Message is expired, but is not logically deleted.

User response: Message expiry occurs when an application attempts to get or browse a message that has expired. At this point, the queue manager attempts to logically delete the message. The queue manager uses several internal transactions and programs to remove expired messages. Ensure that all MQ transactions and programs have been defined and are available to CICS. If maintenance has been recently applied, review any cover letters to ensure special installation instructions have not been missed.

101000W Maximum queue depth reached

Explanation: The maximum queue depth for a given queue has been reached, and a further PUT request has been received.

System action: The PUT request is terminated and the problem queue is marked as "MAX".

User response: Perform one of the following :

1. Process this queue either through an application or the queue maintenance facility.
2. Increase the maximum queue depth for the queue.

101010E Queue concurrent update has occurred

Explanation: Two or more update requests were being received at one time for the same QSN record.

System action: The first request is served while the subsequent requests, deemed concurrent, are rejected.

User response:

1. Review all terminated requests.
2. Re-execute any legitimate requests.

101015W Queue not found

Explanation: An attempt to stop/start a queue has failed because the identified queue does not exist.

System action: The request is terminated unsuccessfully.

User response: Check the queue name matches a queue defined to the queue manager.

101090I Operation rejected for stopped queue

Explanation: A request has been executed against a STOPPED queue.

System action: Terminate the request.

User response: START the problem queue.

101091I Queue disabled and possible operation failure

Explanation: The queue manager flagged a queue disabled due to errors during system initialization, or the queue has been flagged disabled due to an operational failure.

System action: The problem queue is marked STOPPED.

User response:

1. Examine queue definition and file allocation for problems.
2. Check for related error messages in the system log.
3. Correct the queue definition, or resolve associated error messages.

102090C Queue QSN number limit has been reached

Explanation: Queue messages are internally organized with an incremental numeric key field called a Queue Sequence Number (QSN). Being a signed, 4-byte binary field, the QSN is limited to 2,147,483,650 messages per queue.

System action: The PUT request for this queue is rejected.

User response: The queue requires reorganization.

1. An offline reorganization can be achieved using the MQ batch utility job MQPREORG.
2. An online reorganization can be achieved using the automatic reorganization feature.

102091E No space available for PUT request

Explanation: Queue encounters NOSPACE condition for a PUT request.

System action: Terminate the request and mark Queue "FULL".

User response: Perform one of the following :

1. Do file maintenance on this problem queue such as running the batch job MQPREORG.
2. Execute on/line QUEUE Maintenance to delete messages via "Delete by Date/time".

102092E No space available for non-PUT request

Explanation: Queue encounters errors for an UPDATE request, NOSPACE condition occurred.

System action: Terminate the request and mark queue "FULL".

User response: Perform one of the following :

1. Do file maintenance on this problem queue such as running the batch job MQPREORG.
2. Execute on/line QUEUE Maintenance to delete messages via "Delete by Date/time".

104021E Dual queue unavailable

Explanation: Dual destination queue has been STOPPED or was not initialized properly.

System action: Marked Dual Queue as "recovery needed".

User response: Perform one of the following :

1. Try to re-START the Dual Queue.
2. Examine and fix the Queue and File definition for this queue. Refresh Queue or reinitialize system.

104022E Dual queue file error

Explanation: Dual destination Queue had file error or was not initialized properly.

System action: Marked Dual Queue as "recovery needed".

User response: Perform one of the following:

1. Try to re-START the Dual Queue.
2. Examine and fix the Queue and File definition for this queue. Refresh Queue or reinitialize system.

104023E Dual queue out of sequence

Explanation: Dual destination Queue does not match Source Queue.

System action: Marked Dual Queue as "recovery needed".

User response: Examine and fix the Queue and File definition for this queue. Refresh Queue or reinitialize system.

105090E Trigger start by queue start/stop failed

Explanation: MQPSSQ, a subroutine to start / stop a Queue, encounters error to start MQ02, a transaction that handles trigger function.

System action: The request is terminated unsuccessfully.

User response: Examine CICS tables to fix the problem.

105091E Trigger initialization failed due to erroneous data

Explanation: MQPAIP2, a program handling trigger function, receives erroneous data and cannot fulfill the request.

System action: The request is terminated unsuccessfully.

User response: Contact support for WMQ for z/VSE.

106000E Insufficient storage for MQ API exit processing

Explanation: An attempt to allocate storage for WMQ API exits failed.

System action: WMQ API exits are disabled for the connection.

User response: Increase above the line storage resources for the CICS region.

501068E Channel exit definition missing

Explanation: During channel activation, the channel is defined as having exit parameters defined, but these definitions cannot be found.

System action: Channel is terminated.

User response: Check the channel definition and it's exit parameters.

109000E Action not authorized

Explanation: NOAUTH condition flagged by CICS when a resource security check has failed, or MQ security is active and an operation was attempted for an unauthorized user.

System action: The request is terminated unsuccessfully.

User response: Review security mechanism and logs.

300000E Action not supported

Explanation: Start/stop queue request started with invalid data.

System action: Terminate the request.

User response: Review application for call format and data.

300010E Program started incorrectly

Explanation: An MQ module has been started incorrectly.

System action: MQ module terminates.

User response: Check related MQ documentation to ensure the correct data and data format is being passed to the MQ module.

300020E Master terminal or derivative incurred map failure

Explanation: MAPFAIL condition raised in Master Terminal panels (MQMT and its derivatives).

System action: Terminate the request.

User response: Review PPT for MAP modules (MQM????) and fix the problem.

300030C Queue lock table is full

Explanation: Not enough queue lock entries present to insert a new entry.

System action: Terminate the request.

User response: Review application for multiple message retrieval without a SYNCPOINT. If no application problem is present then increase queue lock count to higher value. Note this value is used to calculate an incore table, so caution should be used.

301000C Expected record is missing

Explanation: An expected message was found missing. This is normally occurs under a Delete request.

System action: Terminate the request.

User response: Restart the application.

301010E Duplicate record detected

Explanation: An duplicate message was found. This is normally occurs under a PUT condition.

System action: Terminate the request.

User response: Restart the application.

309010E Queue checkpoint record missing

Explanation: A checkpoint of a Queue was requested and no checkpoint record was found on this queue.

System action: Terminate the request.

User response: Re-initialize system and restart the application.

400000E Internal operation not recognized

Explanation: A call to an MQ module requests an operation that is not recognized.

System action: Terminate the request.

User response: If the MQ module is called by an application, check that the correct data and data format is passed to the MQ module. Otherwise, check that WebSphere MQ has been installed correctly.

400001E Program started with incorrect data length

Explanation: A call to an MQ module passed data that did not match the expected length.

System action: Terminate the request.

User response: If the MQ module is called by an application, check that the correct data and data format is passed to the MQ module. Otherwise, check that WebSphere MQ has been installed correctly, or, if maintenance has been recently applied, check that the maintenance has been applied correctly.

400002E Program started with incorrect data

Explanation: A call to an MQ module passed data that was not recognized, or was not in an expected format.

System action: Terminate the request.

User response: If the MQ module is called by an application, check that the correct data and data format is passed to the MQ module. Otherwise, check that WebSphere MQ has been installed correctly, or, if maintenance has been recently applied, check that the maintenance has been applied correctly.

400003E Internal call generated bad return code

Explanation: A call to an MQ internal module produced an unexpected return code.

System action: Terminate the request.

User response: Check the System Log for related error messages. Also check the z/VSE console and CICS logs for possible related error messages.

400010E Internal data corruption

Explanation: Internal MOVE of data has found corrupt data.

System action: Terminate the request.

User response: Examine any prior messages for actual problem.

402000C Internal data structure missing

Explanation: An internal data structure was not initialized or has been deleted.

System action: Terminate the request.

User response: Check that the queue manager has not been stopped while MQ applications are active. If the queue manager is active, stop and restart the system. If this problem persists, contact WMQ for z/VSE support.

402090C Internal data structure invalid

Explanation: An internal data structure did not contain data as expected.

System action: Terminate the request.

User response: For an application program, check that a valid HCONN parameter is passed in all MQI calls. Otherwise, stop and restart the queue manager. If this problem persists, contact WMQ for z/VSE support.

501001E Channel free error

Explanation: (Reserved)

501002E Channel EIB error

Explanation: (Reserved)

501003E Channel closed by remote MCA

Explanation: The remote MCA closed the channel due to an error on the remote system.

System action: Fatal error - Communication is terminated.

User response: This message indicates the general cause of the channel closure. Error logs on the remote system should be examined for additional information. When the cause has been rectified, the channel can be restarted.

501004E Channel allocation error

Explanation: (Reserved)

501005E Channel allocate retry error

Explanation: (Reserved)

501006E Channel connect error

Explanation: An attempt to connect a channel failed.

System action: Fatal error - Communication is terminated.

User response: Check that the channel being connected is correctly defined on both the local and remote MQ systems, and that the channel is defined with the correct channel type.

501008E Channel send error

Explanation: RECEIVER issued a SEND command and its

System action: Fatal error - Communication is terminated.

User response:

1. Review System Log or error TD Queue for messages prior to this message. TRM in the error message contains the EIBTRMID which is the principal facility associated with this error. Locate any messages associated with this principal facility.
2. Review the EIBRCODE, EIBRESP, and EIBRESP2 fields in the detail portion of the message. They contain information about the cause of the problem. Refer to the CICS/ESA Application Programming Reference manual for an explanation of these values.
3. Correct problem and restart communication.

501009E Receiver responded with error

Explanation: The Sender MCA received an error response from a remote Receiver MCA.

System action: Fatal error - Communication is terminated.

User response: Review the error reason code to determine the reason of the error response and restart the communication after correction. Also check error logs on remote system for additional information.

501010E Channel issued invalid response type

Explanation: Unsupported Message Segment Type received.

System action: Fatal error - Communication is terminated.

User response: Review the Segment type and restart communication without the problem type.

501011E Channel response MSN error

Explanation: (Reserved)

501012E Channel response indicates fatal error

Explanation: (Reserved)

501013I Channel re-negotiation

Explanation: The Receiver MCA has rejected a channel parameter during activation, and has requested that the remote MCA provide a different parameter value.

System action: Reject this proposal and continue with channel negotiation.

User response: No action is needed unless remote platform can not provide an acceptable channel parameter. If this happens then the conflicting parameter must be changed on this or the remote platform.

501014W Channel detected unknown integer encoding

Explanation: The Receiver MCA received an integer encoding parameter of an unknown type.

System action: Channel communication continues.

User response: Check for error messages on the remote system and take action as necessary.

501015W Channel received invalid transmission header

Explanation: The Message Channel Agent received data that was not prepended with expected MCA data structures.

System action: Channel communication continues.

User response: Check for errors on the remote system and take action as necessary.

501016W Unsupported Coded Character Set ID (CCSID)

Explanation: The Message Channel Agent received a request to use a CCSID that is unknown to the queue manager.

System action: Remote system can try a different CCSID or terminate the channel.

User response: Define the CCSID to the local queue manager, or configure the remote system to use a known CCSID.

501017W Channel received invalid message segment header

Explanation: The Receiver MCA received data that did not have the embedded data structures that it expected.

System action: Channel communication is terminated.

User response: Check for errors on the remote system and take action as necessary.

501018W Channel received invalid transmission queue header

Explanation: The Receiver MCA received data that did not have the embedded data structures that it expected.

System action: Channel communication is terminated.

User response: Check for errors on the remote system and take action as necessary.

501019W Channel initiation failed

Explanation: The Receiver MCA could not initiate a channel.

System action: Channel communication is terminated.

User response: Check for errors on the remote system and take action as necessary.

501020W Unsupported FAP level

Explanation: In establishing a channel connection, a remote MQ system is attempting to use a protocol level that is not recognized by the local queue manager.

System action: Channel activation continues.

User response: The channel activation process should negotiate a protocol level that is acceptable to both queue managers.

501021W Message size too large

Explanation: During channel negotiation, a maximum message size was requested, but was too large for one of the queue managers.

System action: Channel negotiation continues.

User response: If channel activation fails, change the channel definition to use a smaller maximum message size.

501022W Message wrap mismatch

Explanation: During channel negotiation, a wrap sequence number was requested, but did not match the wrap value in the associated channel definition.

System action: Channel negotiation continues.

User response: If channel activation fails, ensure that

sender and receiver channel pairs use the same wrap sequence value.

501023E Undeliverable message lost

Explanation: An attempt to place an undeliverable message to the system dead letter queue failed because the dead letter queue was unavailable or the queue manager was inactive.

System action: Process is terminated.

User response: Check that the system dead letter queue is correctly defined and available. Determine the intended target queue for the message from the channel status, and verify that the target queue is correct and available. Recreate the lost message if necessary.

501024E Queue manager unavailable for communication

Explanation: A channel cannot be established because one of the queue managers is unavailable.

System action: Process or communication is terminated.

User response:

1. Check local queue manager and if necessary initialize using MQIT or MQMT.
2. Check the queue manager is active on the remote system.

501025E Unknown inbound channel name

Explanation: The communication cannot be established because the channel name received from the remote system is not defined locally.

System action: This communication session is terminated.

User response: Check the channel name to see if it is correct. Define this in the local definitions or correct the remote system as necessary.

501026E Unknown outbound channel name

Explanation: The channel specified is not available on the remote platform.

System action: This communication session is terminated.

User response: Check on the remote system to see why the channel is not available and define it if necessary.

501027E Channel busy or stopped

Explanation: An attempt was made to use a channel that is already in use, or has been stopped.

System action: Fatal error - Communication is terminated.

User response:

1. Retry the channel later.
2. Start the channel for immediate use.

501028W Channel re-synchronization error

Explanation: During channel activation, the Receiver MCA detected that the requested message sequence number did not match the expected value.

System action: Channel activation continues.

User response: If the channel will not start, verify that expected messages have been sent/received, and manually reset the MSN for both the sender and the receiver channels.

501029E Channel unavailable for use

Explanation: An attempt was made to use a channel that is currently unavailable.

System action: The MCA is terminated without further action.

User response: Check the Enable flag in the channel definition and ensure that it is set to 'Y'. Enable the channel or disable the transmission to prevent unnecessary log messages.

501030E Transmission length error

Explanation: The Receiver MCA has detected a mismatch between an explicit data length and the length of data received over an active connection.

1. The length of the application portion of the message specified in the header exceeds the maximum length defined for this channel.
2. The length of the application portion of the message received is not equal to the length specified in the header.

System action: Fatal error - Communication is terminated.

User response: For reason #1:

1. Review the MAX STA LEN and the MES LEN in the detail portion of the message.
2. Check the configuration of the Receiver channel to insure the maximum message size is set up correctly.
3. Check the configuration of the Sender.
4. Reconfigure if necessary and restart communication.

For reason #2:

1. Review the HEAD MES LEN and the REC LEN in the detail portion of the message.

2. Proper running should preclude this occurrence. Investigate sender/server process for program error.
3. Correct problem and restart communication.

501031W Messages per batch too large

Explanation: During channel negotiation, a messages per batch value was requested, but was too large for one of the queue managers.

System action: Channel negotiation continues.

User response: If channel activation fails, change the channel definition to use a smaller batch size.

501032W Maximum transmission size too large

Explanation: During channel negotiation, a maximum transmission size was requested, but was too large for one of the queue managers.

System action: Channel negotiation continues.

User response: If channel activation fails, change the channel definition to use a smaller transmission size.

501050I Message sequence number has been reset

Explanation: During channel activation, the Receiver MCA has detected that the message sequence number has been manually reset.

System action: Channel negotiation continues if the MSN is valid.

User response: If the channel does not start, manually reset the channel MSN to match the remote system. Care should be taken that expected messages have been transmitted, and transmitted messages are not duplicated by being resent.

501060E Unexpected or invalid channel security exit exchange

Explanation: During channel activation, a channel security exit has been activated that returned an error return code or did not receive an expected security exchange from the remote system.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Ensure that the security requirements for the channel are met before restarting the channel.

501061E Channel security exit program missing or unavailable

Explanation: During channel activation, the initialization of a channel security exit program failed because it was unavailable to the CICS region.

System action: Channel is terminated.

User response: Ensure that the correct exit program is specified in the channel definition. Check that the program is defined to CICS and enabled.

501062E Unexpected or invalid channel message exit response

Explanation: During channel processing, a channel message exit returned an invalid response, or a request to terminate the channel.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Determine the reason for the message exit response and correct.

501063E Channel message exit program missing or unavailable

Explanation: During channel activation, the initialization of a channel message exit program failed because it was unavailable to the CICS region.

System action: Channel is terminated.

User response: Ensure that the correct exit program is specified in the channel definition. Check that the program is defined to CICS and enabled.

501064E Unexpected or invalid channel send exit response

Explanation: During channel processing, a channel send exit returned an invalid response, or a request to terminate the channel.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Determine the reason for the send exit response and correct.

501065E Channel send exit program missing or unavailable

Explanation: During channel activation, the initialization of a channel send exit program failed because it was unavailable to the CICS region.

System action: Channel is terminated.

User response: Ensure that the correct exit program is specified in the channel definition. Check that the program is defined to CICS and enabled.

501066E Unexpected or invalid channel receive exit response

Explanation: During channel processing, a channel receive exit returned an invalid response, or a request to terminate the channel.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Determine the reason for the receive exit response and correct.

501067E Channel receive exit program missing or unavailable

Explanation: During channel activation, the initialization of a channel receive exit program failed because it was unavailable to the CICS region.

System action: Channel is terminated.

User response: Ensure that the correct exit program is specified in the channel definition. Check that the program is defined to CICS and enabled.

501070E Channel exit returned invalid data length

Explanation: A channel exit returned a data length greater than the maximum transmission length.

System action: Channel is terminated.

User response: Check channel exit logic and ensure that the data length returned by the exit does not exceed the maximum transmission length.

501071E Channel exit returned bad exit buffer address

Explanation: A channel exit returned a response code to use an exit buffer, but the exit buffer address was not valid.

System action: Channel is terminated.

User response: Check channel exit logic and ensure that the exit buffer address is valid when the response code indicates its use is required.

501072E Channel exit modified MCA transmission header

Explanation: A send or receive channel exit modified the first 8 bytes of the transmission data.

System action: Channel is terminated.

User response: Check channel exit logic and ensure that the exit does not modify the first 8 bytes of the transmission data.

501073E Unexpected or invalid channel exit response

Explanation: During channel processing, a channel exit returned an invalid response, or a request to terminate the channel.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Determine the reason for the message exit response and correct.

501074E Channel exit error

Explanation: During channel processing, a channel exit returned an invalid response, or an invalid or unexpected data exchange occurred.

System action: Channel is terminated.

User response: Ensure that the correct exit programs are configured to run at either end of the channel. Determine the reason for the exit error and correct.

501075E Channel exit data length error

Explanation: During channel processing, a channel exit indicated that data was available for transmission, but the data length was zero.

System action: Channel is terminated.

User response: Ensure that the exit program sets the data length in accordance with data to be transmitted.

501076W Channel terminated by exit

Explanation: During channel processing, a channel exit indicated that the channel should be closed.

System action: Channel is terminated.

User response: Check that the exits associated with the channel are operating correctly. If so, determine the reason for the exit response and correct.

501077E Security exchange not received

Explanation: During channel processing, a channel security exit expected a security exchange but did not receive one.

System action: Channel is terminated.

User response: Check that the exits associated with the channel are operating correctly. If so, determine the reason for the exit response and correct.

501078E **Channel exit program missing or unavailable**

Explanation: During channel processing, the activation of a channel exit program failed because it was unavailable to the CICS region.

System action: Channel is terminated.

User response: Ensure that the correct exit program is specified in the channel definition. Check that the program is defined to CICS and enabled.

501080E **Channel defined with invalid transmission queue**

Explanation: An attempt to start a sender or server channel failed because the channel definition does not name a valid transmission queue.

System action: Channel is terminated.

User response: Correct the channel definition so that it names a valid transmission queue.

600001C **Prog: xxxxxxxx Error detected. Contact Support.**

Explanation: CICS has detected an error condition not handled by a specific routine.

Operator action: Report to IBM

System action: The dialog was ended.

600005C **Prog: xxxxxxxx ABEND Code zzzz Contact Support.**

Explanation: The program ended due to a CICS problem and the ABEND code *zzzz* was returned to a HANDLE ABEND routine.

Operator action: Report to IBM

System action: The dialog was ended.

600007C **Prog: xxxxxxxx File: yyyyyyy Not Found. Contact Support.**

Explanation: A request has been issued against the file *yyyyyyyy*, but it was not defined in the FCT

Operator action: Contact your system administrator and check whether all WebSphere MQ files were defined in the CICS File Control Table (FCT), and physically allocated by VSAM.

System action: The dialog was ended.

600009C **Prog: xxxxxxxx File: yyyyyyy DISABLED. Contact Support.**

Explanation: CICS tried to access the file *yyyyyyy* which was not enabled.

Operator action: Use “CEMT S DATA” to set the file ENABLED. If the DISABLED status persists, check with the System Administrator.

System action: The dialog was ended.

600011C **Prog: xxxxxxxx File: yyyyyyy ILLOGIC error. Contact Support.**

Explanation: Usually this is related to file input and output. This condition is returned by CICS when the error does not fall within one of the other CICS response categories.

Operator action: Report to IBM

System action: The dialog was ended.

600017C **Prog: xxxxxxxx File: yyyyyyy I/O error. Contact Support.**

Explanation: Normally this is due to hardware errors.

Operator action: Check the System console for more details.

System action: The dialog was ended.

600019C **Prog: xxxxxxxx File: yyyyyyy Record not found. Contact Support.**

Explanation: The program tried to read a record but the request failed.

Operator action: Report to IBM.

System action: The dialog was ended.

600021C **Prog: xxxxxxxx File: yyyyyyy is not open. Contact Support.**

Explanation: CICS tried to access a file which had not been opened, and was unable to open it. This can happen when the file is already in use by another partition.

Operator action: Use “CEMT I DATA” and try to open it manually.

System action: The dialog was ended.

600023C **Prog: xxxxxxxx INVREQ error Contact Support.**

Explanation: A request was received by CICS and cannot be processed for various reasons.

Operator action: Report to IBM

System action: The dialog was ended.

600025C **Prog: xxxxxxxx MAPFAIL error Contact Support.**

Explanation: CICS was unable to display a BMS map on the terminal.

Operator action: Report to IBM

System action: The dialog was ended.

600027C **Prog: xxxxxxxx TRANSID error Contact Support.**

Explanation: WebSphere MQ tried to initiate a transaction, but this transaction was not found in the CICS tables.

Operator action: This is probably an installation error. Check whether the WebSphere MQ group has been correctly installed in the DFHCSD file, and activated. Use CEMT I TRAN(MQ*) to verify this. If everything appears to be correct, report the problem to IBM.

System action: The dialog was ended.

800000E **Unexpected CICS condition**

Explanation: An MQ module trapped a generically handled error condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason.

800010E **Unexpected CICS INVREQ condition raised**

Explanation: An MQ module trapped a CICS INVREQ condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason.

800011C **Unexpected CICS ILLOGIC condition raised**

Explanation: An MQ module trapped a CICS ILLOGIC condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason.

800090E **Checkpoint processing failed**

Explanation: A general error occurred while processing the checkpoint record of a Queue file.

System action: Terminate the request.

User response: Use LISTCAT to review the VSAM file containing this Queue file.

800099E **CICS abnormal end condition**

Explanation: An MQ module trapped a CICS ABEND condition.

System action: Terminate the request.

User response: Perform normal CICS application abend analysis to determine the cause of the abend.

801012E **Unexpected CICS NOTOPEN condition raised**

Explanation: An MQ module trapped a CICS NOTOPEN condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason.

801019E **Unexpected CICS DISABLE condition raised**

Explanation: An MQ module trapped a CICS DISABLE condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason.

802000C **Unexpected CICS NOSTG condition raised**

Explanation: An MQ module trapped a CICS NOSTG condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason, and

1. check that you have not overestimated the maximum size of messages in queue and channel definitions
2. if you have large message sizes then increase the amount of 31-bit storage in the partition that CICS is running

---

803001C **Unexpected CICS LENGERR condition raised**

Explanation: An MQ module trapped a CICS LENGERR condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly.

808000C **Unexpected CICS MAPFAIL condition raised**

Explanation: An MQ module trapped a CICS MAPFAIL condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly.

809000C **Unexpected CICS PGMIDERR condition raised**

Explanation: An MQ module trapped a CICS PGMIDERR condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly. If the error is caused by a trigger event, check that the trigger program is defined to CICS.

809010C **Unexpected CICS FILEID condition raised**

Explanation: An MQ module trapped a CICS FILEID condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly.

809011C **Unexpected CICS FILENOTFOUND condition raised**

Explanation: An MQ module trapped a CICS FILENOTFOUND condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly.

809012C **Unexpected CICS IOERR condition raised**

Explanation: An MQ module trapped a CICS IOERR condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly.

809050C **Unexpected CICS TRANIDERR condition raised**

Explanation: An MQ module trapped a CICS TRANIDERR condition.

System action: Terminate the request.

User response: Produce a CICS auxiliary trace to determine the CICS command failure and the error reason. Check that MQ is installed correctly and that recent maintenance has been applied correctly. If the error is caused by a trigger event, check that the trigger transaction and its associated program are defined to CICS.

900000C **Queue manager environment missing or invalid**

Explanation: An attempt to use queue manager services was made without the system environment being established.

System action: Terminate the request.

User response: Run transaction MQSE to establish the queue manager environment. If the environment exists, check that MQ is installed correctly and that recent maintenance has been applied correctly.

Console Messages

The following messages may be generated by WebSphere MQ and displayed on the z/VSE console during normal operation.

MQI0010I **WMQ for z/VSE environment initializing.**

MQI0037W **CICS SIT parameters may restrict number of WMQ MCA or WMQ Client connections.**

Batch Interface Console Messages

The following message may be generated by the WebSphere MQ batch interface, or WebSphere MQ batch applications. For all messages, the symbol 'mqintfid' is replaced by the batch interface identifier of the queue manager.

Messages MQB0001E to MQB0013E come from WebSphere MQ batch applications. Messages MQI0100I to MQI0110W come from the WebSphere MQ batch interface.

MQB0001E MQS Batch Interface (mqintfid) XPCC error RETC(rc) REAS(re)

Explanation: An XPCC operation failed when an MQI batch application issued an MQI call. The XPCC call issued returned code 'rc' and reason code 're'.

Operator response: Check that the WebSphere MQ batch interface is running in the CICS region that hosts WebSphere MQ. Review z/VSE System Macro documentation for descriptions of the XPCC return and reason codes.

MQB0008E MQS Batch Interface (mqintfid) XPCC SENDR failed

Explanation: An XPCC SENDR operation failed. This message should be accompanied by message MQB0001E which describes the XPCC return and reasons codes.

Operator response: Examine MQB0001E message and follow operator action instructions.

MQB0009E MQS Batch Interface (mqintfid) XPCC CONNECT failed

Explanation: An XPCC CONNECT operation failed. This message should be accompanied by message MQB0001E which describes the XPCC return and reasons codes.

Operator response: Examine MQB0001E message and follow operator action instructions.

MQB0010E MQS Batch Interface (mqintfid) XPCC IDENTIFY failed

Explanation: An XPCC IDENTIFY operation failed. This message should be accompanied by message MQB0001E which describes the XPCC return and reasons codes.

Operator response: Examine MQB0001E message and follow operator action instructions.

MQB0013E MQS Batch Interface (mqintfid) GETVIS failed

Explanation: An attempt to obtain GETVIS storage failed.

Operator response: Increase the available GETVIS storage for the batch partition.

MQI0100I MQS Batch Interface (mqintfid) started.

Explanation: The WebSphere MQ batch interface has started.

Operator response: None.

MQI0101E MQS Batch Interface (mqintfid) GETVIS failed

Explanation: The WebSphere MQ batch interface failed to obtain sufficient GETVIS storage to start the batch interface. Batch interface cannot be started.

Operator response: Increase the amount of 31 bit GETVIS storage resources available to the CICS region.

MQI0102I MQS Batch Interface (mqintfid) stop requested

Explanation: A request to stop the WebSphere MQ batch interface has been registered.

Operator response: None.

MQI0103E MQS Batch Interface (mqintfid) userid error

Explanation: The WebSphere MQ batch interface cannot start a mirror transaction (MQBX) for a batch program because the user associated with the interface is not a surrogate for the batch user, or the user is invalid.

Operator response: Check that the batch job includes a valid // ID card that identifies a valid userid and password, and that the interface user is a surrogate for that userid.

MQI0104I MQS Batch Interface (mqintfid) ended

Explanation: The WebSphere MQ batch interface has stopped.

Operator response: None.

MQI0105E MQS Batch Interface (mqintfid) abending

Explanation: The WebSphere MQ batch interface has encountered an abend condition and is terminating.

Operator response: Wait for the interface to abend,

check accompanying console messages. Examine relevant dumps.

MQI0107E MQS Batch Interface (mqintfid) init failed

Explanation: The WebSphere MQ batch interface could not establish an XPCC identity for batch connections.

Operator response: Check that the queue manager is configured with a valid batch interface identifier.

MQI0108I MQS Batch Interface (mqintfid) system inactive

Explanation: An attempt to start the WebSphere MQ batch interface was made when the WebSphere MQ queue manager was inactive.

Operator response: Start the WebSphere MQ queue manager before starting the batch interface.

MQI0109I MQS Batch Interface (mqintfid) not started

Explanation: An attempt to stop the WebSphere MQ batch interface was made when the interface was not active.

Operator response: None.

MQI0110W MQS Batch Interface (mqintfid) GETVIS failed

Explanation: The WebSphere MQ batch interface failed to obtain sufficient GETVIS storage to handle the next batch job connecting while current batch job(s) are active.

Operator response: Increase the amount of 31 bit GETVIS storage resources available to the CICS region.

Automatic reorganization console messages

Messages prefixed by "MQPVSAM:" are only displayed when an error occurs. Refer to the MQ SYSTEM.LOG for more details.

The WMQ for z/VSE automatic reorganization feature can also generate the follow message:

MQPIDCMS Insufficient below line GETVIS for reorg

Explanation: There is not enough GETVIS-24 to perform the automatic reorganization.

Operator action: Refer to CICS system programmer to allocate more GETVIS-24 to this partition.

System action: The automatic reorganization is not performed.

Appendix H. Security implementation

This appendix provides a sample security configuration. The sample includes configuration for:

- WebSphere MQ datasets
- WebSphere MQ transactions
- WebSphere MQ system users
- Application users
- Connections
- Queues
- Namelists
- Batch users
- Clients
- Commands
- Command resources
- WebSphere MQ startup
- WebSphere MQ shutdown

The examples in this appendix use CA-Top Secret as the External Security Manager (ESM), or the Basic Security Manager (BSM) when the queue manager is running on z/VSE 4.3 or later system.

Note: The examples in this appendix are only a sample security configuration, and are not intended to define how you should secure your VSE or CICS TS systems.

Before you install

Before installation, you need to make several decisions regarding security. The first is whether to install security or not. If you do want to install security, as this appendix assumes, you need to:

- Modify the MQCICDCT sample JCL.
- Modify the SYSIN installation parameter file.

These two steps are described in Chapter 2, “Installation,” on page 13.

The MQCICDCT.A sample DCT definition file must be modified if you want a user other than the CICS default user to log messages to the SYSTEM.LOG or manage message expiry. Remember that logging messages requires CONNECT and QUEUE authority, and message expiry involves CONNECT and authority to put messages to reply queues.

In addition, the generation of instrumentation events requires that the IE processor transaction triggered by requests arriving on the MQIE transient data queue require CONNECT and OPEN access for the event queues.

This example uses a user (MQADMUSR) other than the CICS default user. The DCT definitions should be changed as follows:

```
MQER     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQER,
                DESTFAC=FILE,
                USERID=MQADMUSR,  <--- Note insertion
                TRANSID=MQER,
```

```
                TRIGLEV=1
MQXP     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQXP,
                DESTFAC=FILE,
                USERID=MQADMUSR,  <---Note insertion
                TRANSID=MQXP,
                TRIGLEV=1
MQIE     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQIE,
                DESTFAC=FILE,
                USERID=MQADMUSR,   <---Note insertion
                TRANSID=MQIE,
                TRIGLEV=1
MQAC     DFHDCT TYPE=INTRA,
                RSL=PUBLIC,
                DESTID=MQAC,
                DESTFAC=FILE,
                USERID=MQADMUSR,   <--- Note insertion
                TRANSID=MQAC,
                TRIGLEV=1
```

If using BSM, these surrogate definitions are required where *cicsjob* is the userid on the CICS job // ID statement:

```
ADD    SURROGAT MQADMUSR.DFHINSTL      UACC(NONE)
PERMIT SURROGAT MQADMUSR.DFHINSTL      ID(cicsjob)  ACCESS(READ)
```

In addition, this surrogate should be defined for the default CICS userid (CICSUSER in example below):

```
ADD    SURROGAT CICSUSER.DFHINSTL      UACC(NONE)
PERMIT SURROGAT CICSUSER.DFHINSTL      ID(cicsjob)  ACCESS(READ)
```

The SYSIN.Z installation configuration file contains default settings for security, where the default is DISABLED. You must change this default to ENABLED before installation. However, you cannot do this until you have installed the WebSphere MQ library from tape. With the PTF for PM02556 applied, you can either edit the SYSIN.Z member or simply use the new SET input statement to override the data from the SYSIN.Z member which follows the SET input.

The relevant entries in the SYSIN parameter file follow the QMDEF heading and should be changed from:

```
QM-STATUS-SECURITY       DISABLED
QM-AUDIT-SECURITY        DISABLED
```

To:

```
QM-STATUS-SECURITY       ENABLED
QM-AUDIT-SECURITY        ENABLED
```

Note that the AUDIT parameter is not implemented in Version 3.0.0, but is reserved for future expansion. It is enabled in this example for consistency.

The configuration file can either be updated by running the MQJSETUP job and then the CICS transaction MQSU, or by using the UPDATE function of the batch MQPUTIL program. See example below.

In addition, WebSphere MQ for z/VSE security uses the queue manager's subsystem identifier (SSID) when building resource profiles to check permissions. For example:

```
ssid.resource_name
```

The SSID is also listed as one of the QMDEF default values. Specifically:

```
QM-SUBSYSID MQV1
```

You should set the identifier to a 4-character value that uniquely identifies you queue manager. If you set the SSID to blanks (spaces), the queue manager name is used instead of the SSID when checking resource profiles.

You can change your SSID after installation if necessary by changing the SYSIN.Z installation file to have the QM-SUBSYSID value you require, and then rerun the MQJSETUP.Z job followed by the MQSU transaction in CICS.

Again with the PTF for PM02556 applied, you can also specify SET QM-SUBSYSID *xxxx* before the SYSIN.Z statements to override the default MQV1 value without having to edit the SYSIN.Z member.

For example, the JCL shown here overrides the SYSIN,Z member value in MQJSETUP:

```
// DLBL LOADFL,'wmqzvse.msfsset',,VSAM,CAT=VSESPUC
// EXEC IESVSMLD,SIZE=AUTO
80,E,LOADFL
SET QM-STATUS-SECURITY ENABLED
SET QM-AUDIT-SECURITY ENABLED
* $$ SLI MEM=SYSIN.Z,S=prd2.wmqzvse
/*
```

External security manager configuration

For WebSphere MQ security to work, certain facilities must be available with your ESM. For CA-Top Secret, you need to ensure these settings exist in the system parameter file:

```
FACILITY(CICSPROD=MODE=FAIL)
FACILITY(CICSPROD=FACMATRX=YES)
FACILITY(CICSPROD=EXTSEC=YES)
FACILITY(CICSPROD=XFCT=YES)
FACILITY(CICSPROD=XDEF)
FACILITY(CICSPROD=XUSER=YES)
FACILITY(CICSPROD=RES)
```

These parameters assume that the facility of the WebSphere MQ CICS region will be CICSPROD. If you are using CICSTEST, or a different facility, you should make the appropriate changes to the parameter file.

The `CICSPROD=RES` parameter ensures the standard WebSphere MQ classes are available. If you change the parameter file, you also need to stop and restart your CA-Top Secret system.

Basic Security Manager (BSM) configuration

If you are running on z/VSE 4.3 or later, then you can use BSM for WebSphere MQ security.

Running following batch BSTADMIN STATUS job

```
// JOB MQBSM000
// ID USER=FORSEC,PWD=FORSEC
// DLBL BSTCNTL,'VSE.BSTCNTL.FILE',,VSAM,CAT=VSESPUC
```

```
// EXEC BSTADMIN,OS390
STATUS
/*
/&
```

reports

```
// JOB MQBSM000
// DLBL BSTCNTL,'VSE.BSTCNTL.FILE',,VSAM,CAT=VSESPUC
// EXEC BSTADMIN,OS390
1S54I  PHASE BSTADMIN IS TO BE FETCHED FROM IJSYSRS.SYSLIB
 STATUS
CLASS     ACTIVE    CMDAUDIT
-----     ------    --------
USER      YES       NO
GROUP     YES       NO
DATASET   YES       NO
VSELIB    YES       NO
VSESLIB   YES       NO
VSEMEM    YES       NO
TCICSTRN  YES       NO
ACICSPCT  YES       NO
DCICSDCT  YES       NO
FCICSFCT  YES       NO
JCICSJCT  YES       NO
MCICSPPT  YES       NO
SCICSTST  YES       NO
APPL      YES       NO
FACILITY  YES       NO
MQADMIN   NO        NO
MQCMDS    NO        NO
MQCONN    NO        NO
MQNLIST   NO        NO
MQQUEUE   NO        NO
SURROGAT  NO        NO
```

You should run a BSTADMIN job to set the WebSphere MQ classes MQADMIN MQCMDS MQCONN MQNLIST MQQUEUE and SURROGAT active.

CICS should also be started with XUSER=YES if SURROGAT class is to be active.

System and application users

This example uses the following system users:

CICSP1
CICS region user

CICSP1DF
CICS default user

MQM Owner of all WebSphere MQ resources

MQADMUSR
WebSphere MQ Startup and Administrative user

The example also uses the following application users:

JOHNS
Application user

JANED
Application user

SHELLYS
Client user

STEVEJ
Batch user

To create these users, you might use the following TSS commands:

```
--  CICS & MQ Departments
TSS CREATE(CICSP1G)  NAME('CICSP1 GROUP')         TYPE(DEPARTMENT)
TSS CREATE(MQ)       NAME('WebSphere MQ 3.0.0 GROUP') TYPE(DEPARTMENT)

--  CICS System Users
TSS CREATE(CICSP1)   NAME('CICSP1 REGION') TYPE(USER) FAC(BATCH)  +
                     DEPT(CICSP1G) PAS(P1CICS,0)                  +
                     MASTFAC(CICSPROD) NORESCHK NOLCFCHK NODSNCHK
TSS CREATE(CICSP1DF) NAME('CICSP1 DEFAULT USER')  TYPE(USER)      +
                     DEPT(CICSP1G) PAS(NOPW,0) FAC(CICSPROD)

--  MQ System Users
TSS CREATE(MQM)      NAME('MQM OWNER')            TYPE(USER)      +
                     DEPT(MQ) PAS(MQSER1ES,0) FAC(CICSPROD)
TSS CREATE(MQADMUSR) NAME('WebSphere MQ ADMIN USER')  TYPE(USER)      +
                     DEPT(CICSP1G) PAS(ADM1N,0) FAC(CICSPROD)

--  MQ Application Users
TSS CREATE(JOHNS)    NAME('JOHN SMITH')           TYPE(USER)      +
                     DEPT(MQ) PAS(SM1TH,0)   FAC(CICSPROD)
TSS CREATE(JANED)    NAME('JANE DOE')             TYPE(USER)      +
                     DEPT(MQ) PAS(D0E,0)     FAC(CICSPROD)
TSS CREATE(SHELLYS)  NAME('SHELLY SIMPSON')       TYPE(USER)      +
                     DEPT(MQ) PAS(SIMPS0N,0) FAC(CICSPROD)
TSS CREATE(STEVEJ)   NAME('STEVEN JONES')         TYPE(USER)      +
                     DEPT(MQ) PAS(J0NES,0)   FAC(BATCH,CICSPROD)
```

For BSM define user with user type Programmer or General. Do not use Administrator as this type is allowed access to all resources. Define a group WMQ and CONNECT the users to this group.

For example, to define a group and connect users to it, use these BSTADMIN commands:

```
ADDGROUP  MQ DATA('WebSphere MQ user')
CONNECT MQ JOHNS
CONNECT MQ JANED
    CONNECT MQ SHELLYS
    CONNECT MQ STEVEJ
* Activate the changes of the BSM control file
    PERFORM DATASPACE REFRESH
```

WebSphere MQ datasets

Before you start up your CICS and WebSphere MQ systems, you should consider WebSphere MQ dataset security. There is little point in protecting WebSphere MQ resources, but then allowing unrestricted access to the WebSphere MQ datasets in which such objects reside.

To protect your datasets in CICS, assuming they use the default names provided with the sample JCL, you could use the following TSS commands:

```
-- Assign ownership
TSS ADD(MQM) FCT(MQFCNFG)
TSS ADD(MQM) FCT(MQFLOG)
TSS ADD(MQM) FCT(MQFMON)
TSS ADD(MQM) FCT(MQFERR)
```

```
TSS ADD(MQM) FCT(MQFREOR)
TSS ADD(MQM) FCT(MQFSSET)
TSS ADD(MQM) FCT(MQFI001)
TSS ADD(MQM) FCT(MQFI002)
TSS ADD(MQM) FCT(MQFI003)
TSS ADD(MQM) FCT(MQFO001)
TSS ADD(MQM) FCT(MQFO002)
TSS ADD(MQM) FCT(MQFO003)

-- Assign permissions to Admin user
TSS PER(MQADMUSR) FCT(MQFCNFG) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFLOG)  ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFMON)  ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFERR)  ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFREOR) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFSSET) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFI001) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFI002) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFI003) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFO001) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFO002) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFO003) ACC(ALL)
```

The equivalent BSM BSTADMIN commands are:

```
ADD FCICSFCT  MQFSSET  UACC(NONE)
ADD FCICSFCT  MQFCNFG  UACC(NONE)
ADD FCICSFCT  MQFACMD  UACC(NONE)
ADD FCICSFCT  MQFARPY  UACC(NONE)
ADD FCICSFCT  MQFIECE  UACC(NONE)
ADD FCICSFCT  MQFIEPE  UACC(NONE)
ADD FCICSFCT  MQFIEQE  UACC(NONE)
ADD FCICSFCT  MQFERR   UACC(NONE)
ADD FCICSFCT  MQFLOG   UACC(NONE)
ADD FCICSFCT  MQFMON   UACC(NONE)
ADD FCICSFCT  MQFREOR  UACC(NONE)
ADD FCICSFCT  MQFI001  UACC(NONE)
ADD FCICSFCT  MQFI002  UACC(NONE)
ADD FCICSFCT  MQFI003  UACC(NONE)
ADD FCICSFCT  MQFO001  UACC(NONE)
ADD FCICSFCT  MQFO002  UACC(NONE)
ADD FCICSFCT  MQFO003  UACC(NONE)
```

Set DFHSIT parameter XFCT=YES to protect the files defined above in resource class FCICSFCT.

If the Programmable Command Formats (PCF) and WebSphere MQ Command (MQSC) features are required, the following datasets should also be protected:

```
TSS PER(MQADMUSR) FCT(MQFACMD) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFARPY) ACC(ALL)
```

Similarly, if the Instrumentation Events feature is required, the following datasets should also be protected:

```
TSS PER(MQADMUSR) FCT(MQFIEQE) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFIECE) ACC(ALL)
TSS PER(MQADMUSR) FCT(MQFIEPE) ACC(ALL)
```

At this point, because the facilities matrix for CICSPROD has `XFCT= YES`, no users other than MQM and MQADMUSR have access to the WebSphere MQ datasets under CICS. Appropriate permissions for other users are described later.

You should also protect your datasets outside CICS. To do this, you might use the following:

```
TSS ADD(MQM) DSN(WMQZVSE)
```

This command establishes generic ownership of all datasets that are prefixed with WMQZVSE.

Protecting transactions

The CICSPROD=XDEF facility matrix setting ensures that users cannot run transactions without explicit permission. In this example, you grant full access to the MQADMUSR user, and restricted access to application users:

```
TSS ADD(MQADMUSR) TRANS(CICSPROD,MQ(G),TST(G))
```

By default, with CA-Top Secret, transactions beginning with TS are in the protection bypass list. Because WebSphere MQ uses TST1, TST2, and TST3 transactions, to introduce protection, you should issue the following command:

```
TSS MODIFY FAC(CICSPROD=PROTADD(TRANID=TST))
```

You should also consider protecting programs. In this example, restricting access to transactions in CICS should be sufficient, and the protection of programs is omitted.

Regardless of whether the WebSphere MQ transactions are protected, if security is active, the use of the transactions should be further restricted by command and command resource security.

Command and command resource security not only restrict the use of the WebSphere MQ transactions, but also PCF and MQSC commands.

Resource ownership

The ESM classes that are relevant to WebSphere MQ for z/VSE are:

- MQADMIN
- MQCONN
- MQNLIST
- MQQUEUE
- MQCMDS

Resources defined to these classes must first have ownership. For this example, all such resources will be owned by user MQM. You can assign ownership as follows:

```
TSS ADD(MQM) MQADMIN(MQV1)
TSS ADD(MQM) MQCONN(MQV1)
TSS ADD(MQM) MQQUEUE(MQV1)
TSS ADD(MQM) MQCMDS(MQV1)
TSS ADD(MQM) MQNLIST(MQV1)
```

Because these classes are generic, all resources prefixed MQV1 are owned automatically by user MQM. At this point, no specific resources have been defined. Appropriate user permissions for class resources are described later.

Resource protection

At this point, WebSphere MQ datasets and transactions are protected. You are now ready to grant users access to WebSphere MQ resources. These include:

- Connections
- Queues
- Namelists

This example makes the following assumptions:

- All application users have authority to connect to the WebSphere MQ queue manager.
- Queue TEST.Q1 is defined in file MQFI001.
- Queue TEST.Q2 is defined in file MQFI002.
- Queue TEST.Q3 is defined in file MQFI003.
- Application user JOHNS requires read/write authority to all three application queues.
- Application user JANED requires read/write authority to TEST.Q1 and TEST.Q2.
- Application user SHELLYS is a client user and requires authority to read/write to TEST.Q3.
- Application user STEVEJ is a batch user and requires authority to browse TEST.Q3.

To grant authority to all users to connect to the WebSphere MQ queue manager, issue the following commands:

```
TSS PER(MQADMUSR) MQCONN(MQV1.CICS) ACC(READ)
TSS PER(JOHNS)   MQCONN(MQV1.CICS) ACC(READ)
TSS PER(JANED)   MQCONN(MQV1.CICS) ACC(READ)
TSS PER(SHELLYS) MQCONN(MQV1.CICS) ACC(READ)
TSS PER(STEVEJ)  MQCONN(MQV1.CICS) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
PERMIT  MQCONN MQV1.CICS  UID(MQADMUSR) ACC(READ)
PERMIT  MQCONN MQV1.CICS  UID(JOHNS)    ACC(READ)
PERMIT  MQCONN MQV1.CICS  UID(JANED)    ACC(READ)
PERMIT  MQCONN MQV1.CICS  UID(SHELLYS)  ACC(READ)
PERMIT  MQCONN MQV1.CICS  UID(STEVEJ)   ACC(READ)
PERFORM DATASPACE REFRESH
```

Note that the MQADMUSR user is also the user assigned to the MQER DCT entry. This user must have CONNECT authority and must be able to write to the SYSTEM.LOG queue.

To grant authority to each user to access specific queues, using the assumptions listed earlier, issue the following commands:

```
TSS PER(MQADMUSR) MQQUEUE(MQV1) ACC(ALL)

TSS PER(JOHNS)   MQQUEUE(MQV1.TEST.Q1) ACC(READ,UPDATE)
TSS PER(JOHNS)   MQQUEUE(MQV1.TEST.Q2) ACC(READ,UPDATE)
TSS PER(JOHNS)   MQQUEUE(MQV1.TEST.Q3) ACC(READ,UPDATE)

TSS PER(JANED)   MQQUEUE(MQV1.TEST.Q1) ACC(READ,UPDATE)
TSS PER(JANED)   MQQUEUE(MQV1.TEST.Q2) ACC(READ,UPDATE)

TSS PER(SHELLYS) MQQUEUE(MQV1.TEST.Q3) ACC(READ,UPDATE)

TSS PER(STEVEJ)  MQQUEUE(MQV1.TEST.Q3) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
PERMIT  MQQUEUE 'MQV1'    UID(MQADMUSR) ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q1'  UID(JOHNS)  ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q2'  UID(JOHNS)  ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q3'  UID(JOHNS)  ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q1'  UID(JANED)  ACC(UPDATE)
```

```
PERMIT  MQQUEUE 'MQV1.TEST.Q2'  UID(JANED)  ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q3'  UID(SHELLYS)  ACC(UPDATE)
PERMIT  MQQUEUE 'MQV1.TEST.Q3'  UID(STEVEJ) ACC(READ)
PERFORM DATASPACE REFRESH
```

Access to underlying WebSphere MQ datasets must also be granted. Following the listed assumptions, you need to issue the following commands:

```
TSS PER(JOHNS)   FCT(MQFI001) ACC(INQUIRE,READ,WRITE)
TSS PER(JOHNS)   FCT(MQFI002) ACC(INQUIRE,READ,WRITE)
TSS PER(JOHNS)   FCT(MQFI003) ACC(INQUIRE,READ,WRITE)

TSS PER(JANED)   FCT(MQFI001) ACC(INQUIRE,READ,WRITE)
TSS PER(JANED)   FCT(MQFI002) ACC(INQUIRE,READ,WRITE)

TSS PER(SHELLYS) FCT(MQFI003) ACC(INQUIRE,READ,WRITE)

TSS PER(STEVEJ)  FCT(MQFI003) ACC(INQUIRE,READ)
```

For testing purposes, you can use the TST2 transaction, which allows users to read and write messages to queues. To allow users to use this transaction, issue the following commands:

```
TSS ADD(JOHNS)   TRANS(CICSPROD,TST2)
TSS ADD(JANED)   TRANS(CICSPROD,TST2)
```

Note that SHELLYS and STEVEJ do not need the TST2 transaction. SHELLYS is a client user and can issue MQI calls directly from a remote MQI client program. STEVEJ is a batch user and similarly can issue MQI calls from a batch partition.

Namelist permissions

Permissions for namelist objects is similar to queues. To grant authority to a user to access a specific namelist, for example user STEVEJ to access namelist A.NAMELIST, issue the following commands:

```
TSS PER(MQADMUSR) MQNLIST(MQV1) ACC(ALL)

TSS PER (STEVEJ) MQNLIST(MQV1.A.NAMELIST) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
PERMIT  MQNLIST 'MQV1'      UID(MQADMUSR) ACC(ALTER)
PERMIT  MQNLIST 'MQV1.A.NAMELIST' UID(STEVEJ) ACC(READ)
PERFORM DATASPACE REFRESH
```

The only permission that is relevant is READ, because namelists can only be opened using the MQOO_INQUIRE open option.

Unlike queues, namelists are not associated with different datasets. Instead, all namelists and their details are stored in the WebSphere MQ configuration file MQFCNFG. Consequently, for STEVEJ to be able to open the namelist, he will also need read access to the WebSphere MQ configuration file. For example:

```
TSS PER (STEVE) FCT(MQFCNFG) ACC(READ)
```

Batch user permissions

Batch users identify themselves to the External Security Manager via the // ID card. For example:

```
// ID USER=STEVEJ,PWD=J0NES
```

Batch user permissions

WebSphere MQ security uses the user name from the ID card and passes it to the WebSphere MQ Batch Interface transaction running under CICS. The user that starts the batch interface must be a surrogate for batch users who want to use the batch interface.

In this example, you use the MQADMUSR user to start the batch interface and act as surrogate to any batch users (that is, STEVEJ). To register MQADMUSR as a surrogate, you can issue the following command:

```
TSS ADD(MQADMUSR) SURROGAT(STEVEJ)
```

Note: For the surrogate feature to be active, the facility matrix option XUSER=YES must be set.

When the batch user attempts to establish a connection to the queue manager, the batch interface user (MQADMUSR) starts the partner transaction (MQBX) as the batch user. This is why the batch interface user must be a surrogate for the batch user.

For BSM, the BSTADMIN commands are:

```
ADD SURROGAT STEVEJ.DFHSTART  UACC(NONE)
PERMIT  SURROGAT STEVEJ.DFHSTART  ID(MQADMUSR) ACCESS(READ)
PERFORM DATASPACE REFRESH
```

From this point on, all WebSphere MQ API calls issued by the batch user will be treated as if they were issued by the batch user under CICS. Therefore, the batch user should be granted the appropriate MQCONN and MQQUEUE privileges.

The batch user also needs authority to execute the MQBX transaction, for example:

```
TSS ADD(STEVEJ) TRANS(CICSPROD,MQBX)
```

Client user permissions

Client users should be treated the same as CICS application users. For security purposes, they are the same. WebSphere MQ API calls issued by client programs are treated as if they were issued by the client user under CICS.

In this example, the client user SHELLYS has already been granted the necessary authority to get and put messages to the TEST.Q3 queue.

Java program clients are a special case for client user permissions. Existing WebSphere MQ Java classes may attempt to open the queue manager as an object during an MQCONN request. This means, for such clients, the client user must have READ access to the queue manager object. For example:

```
TSS PER(SHELLYS) MQQUEUE(MQV1.MQV1) ACC(READ)
```

For BSM, the BSTADMIN command is:

```
PERMIT  MQQUEUE  'MQV1.MQV1'    ID(SHELLYS) ACCESS(READ)
```

Command permissions

Authority to issue commands is required when command or command resource security is active. Command authority is required to create, modify, delete and display WebSphere MQ objects such as the queue manager, channels and queues.

Commands can be issued via the WebSphere MQ master terminal transaction (for example, MQMT), via PCF messages, and the MQSC command utility.

Command permissions involve resources belonging to the MQCMDS class. For a full list of these resources, and the permissions required for each command, refer to section "Resource definitions for command security" on page 666.

Full command authority can be granted to the MQADMUSR user by issuing the following TSS command:

```
TSS PER(MQADMUSR) MQCMDS(MQV1) ACC(ALL)
```

For BSM, the BSTADMIN command is:

```
PERMIT  MQCMDS  MQV1    GEN ID(MQADMUSR)  ACCESS(ALTER)
```

Since all command resources are prefixed with the queue manager name (system identifier), the MQADMUSR user will have ACC(ALL) to any and all of these resources.

Permissions for commands can also be granted by command type or for each individual command. For example, to grant permissions by command type to user JOHNS to issue DISPLAY commands, issue the following:

```
TSS PER(JOHNS) MQCMDS(MQV1.DISPLAY) ACC(READ)
```

For BSM, the BSTADMIN command is:

```
PERMIT  MQCMDS  MQV1.DISPLAY    ID(JOHNS)  ACCESS(READ)
```

User JOHNS can now issue DISPLAY commands to examine the queue manager, channels and queues.Alternatively, to restrict JOHNS to DISPLAY commands for queues only, that is to restrict user JOHNS to individual commands rather than commands by type, issue the following:

```
TSS PER(JOHNS) MQCMDS(MQV1.DISPLAY.QALIAS)  ACC(READ)
TSS PER(JOHNS) MQCMDS(MQV1.DISPLAY.QLOCAL)  ACC(READ)
TSS PER(JOHNS) MQCMDS(MQV1.DISPLAY.QREMOTE) ACC(READ)
```

For BSM, the BSTADMIN commands are:

```
PERMIT  MQCMDS  'MQV1.DISPLAY.QALIAS'     ID(JOHNS)  ACCESS(READ)
PERMIT  MQCMDS  'MQV1.DISPLAY.QLOCAL'     ID(JOHNS)  ACCESS(READ)
PERMIT  MQCMDS  'MQV1.DISPLAY.QREMOTE'    ID(JOHNS)  ACCESS(READ)
```

Note: As no mixed case character or %, _, or / symbols are used in the MQ object names in the above example, the profile name could have been specified without quotes.

User JOHNS can now issue DISPLAY commands for any type of queue, but not for the queue manager or channels.

Command resource permissions

If command resource security is active, command permissions are insufficient to issue commands against specific resources. A user must also be granted command resource permissions to those specific resources.

Command resource security should not be confused with command security. Command security restricts access to commands, whereas command resource

security restricts issuing commands against specific resources. Consequently, command resource security is only relevant for commands that affect specific objects (that is, queues and channels).

Resources for command resource security are defined to the MQADMIN class.

Full command resource authority can be granted the MQADMUSR user by issuing the following TSS command:

```
TSS PER(MQADMUSR) MQADMIN(MQV1) ACC(ALL)
```

For BSM, the BSTADMIN command is:

```
PERMIT  MQADMIN  MQV1   GEN ID(MQADMUSR)  ACCESS(ALTER)
```

Once again, since all command resources are prefixed with the queue manager's SSID, the MQADMUSR user will have ACC(ALL) to any and all of the command resources defined to the MQADMIN class.

Alternatively, since command resource types are limited to channels and queues, the MQADMUSR user could be granted full command resource authority by issuing the following commands:

```
TSS PER(MQADMUSR) MQADMIN(MQV1.CHANNEL) ACC(ALL)
TSS PER(MQADMUSR) MQADMIN(MQV1.QUEUE)   ACC(ALL)
```

For BSM, the BSTADMIN commands are:

```
PERMIT MQADMIN MQV1.CHANNEL   ID(MQADMUSR)  ACCESS(ALTER)
PERMIT MQADMIN MQV1.QUEUE     ID(MQADMUSR)  ACCESS(ALTER)
```

Like command security, permissions can be granted by object type or for each individual object. For example, to grant permissions by command resource type to user JOHNS to issue ALTER commands for any queue object, issue the following:

```
TSS PER(JOHNS) MQADMIN(MQV1.QUEUE) ACC(ALTER)
```

For BSM, the BSTADMIN command is:

```
PERMIT MQADMIN MQV1.QUEUE   ID(JOHNS)  ACCESS(ALTER)
```

User JOHNS can now issue ALTER commands to modify any queue providing he also has command authority to issue ALTER commands (assuming command security is active).

Alternatively, to restrict JOHNS to ALTER commands for queues TEST.Q1 and TEST.Q2, that is to restrict user JOHNS to individual objects rather than commands by type, issue the following:

```
    TSS PER(JOHNS) MQADMIN(MQV1.QUEUE.TEST.Q1) ACC(ALTER)
TSS PER(JOHNS) MQADMIN(MQV1.QUEUE.TEST.Q2) ACC(ALTER)
```

For BSM, the BSTADMIN commands are:

```
PERMIT MQADMIN 'MQV1.QUEUE.TEST.Q1'   ID(JOHNS)  ACCESS(ALTER)
PERMIT MQADMIN 'MQV1.QUEUE.TEST.Q2'   ID(JOHNS)  ACCESS(ALTER)
```

User JOHNS can now issue ALTER commands for TEST.Q1 and TEST.Q2, but not for any other queue.

Command resource permissions are not required for DISPLAY commands.

Trigger permissions

Trigger programs and transactions are started automatically when an application program puts a message to a queue that is defined to start a trigger. The invocation and control of trigger instances is handled by WebSphere MQ transaction MQ02.

Therefore, if an application user puts messages to a queue that may start a trigger instance, that user must have authority to run the MQ02 transaction.

In this example, you could define TEST.Q1 to start a trigger program every time a message is put to the queue. For example, the trigger program may get a message from TEST.Q1 and put a message on TEST.Q2. To enable application user JANED to put messages to TEST.Q1 and successfully start the trigger instance, you need to grant authority to the MQ02 transaction. For example:

```
TSS ADD(JANED) TRANS(CICSPROD,MQ02)
```

If you do not grant this authority, JANED can successfully put messages to the target queue, but the trigger instance will ABEND.

If you trigger a transaction, the application user must also have authority to run the trigger transaction. If you use program security, the application user needs authority to run the trigger program and a range of WebSphere MQ programs (the exact programs are beyond the scope of this Appendix).

If you use the trigger option Allow Restart of Trigger in a queue definition, transaction MQSM will, when appropriate, attempt to run the MQ02 transaction. MQSM runs as the WebSphere MQ startup user. Therefore, for security purposes, you should be careful when using the Allow Restart of Trigger feature. For details about this option, see “Trigger Information” on page 102.

CICS startup

Your CICS startup deck should include a // ID card. For the example user CICSP1, the // ID card would appear as follows:

```
// JOB jobname
// ID USER=CICSP1,PWD=P1CICS
```

You also need to identify the CICS default user as a SIT parameter. For the example user CICSP1DF, the SIT parameter would appear as follows:

```
DFLTUSER=CICSP1DF
```

Starting WebSphere MQ

It is important that WebSphere MQ is started by a user with sufficient authority. In this example, the user MQADMUSR has full access to the WebSphere MQ datasets, transactions, and WebSphere MQ resources, including connection authority.

To start WebSphere MQ, log on to CICS as MQADMUSR and run the following transactions:

- MQSE
- MQIT

Another way to start WebSphere MQ is to run 'MQSE I', or use the MQMT transaction, option 2.4.

A further option is to use the PLTPI program. The DFHPLT macro does not allow you to specify a userid with a PLTPI program. However, you can specify a SIT parameter for PLTPIUSR. For example:

```
PLTPIUSR=PLTUSER
```

In such a case, the PLTPIUSR (that is, PLTUSER) must be authorized to the appropriate resources defined by PLTPISEC.

Remember that your PLTPIUSR may run programs that are not relevant to WebSphere MQ for z/VSE, so, in this example implementation, it may not be appropriate to use user MQADMUSR. Therefore, you can define a special PLTPIUSR as follows:

```
--  Create the PLTPIUSR
TSS CREATE(PLTUSER)  NAME('CICS PLTPI USER')  TYPE(USER)      +
                     DEPT(CICSP1G) PAS(PLTP1,0) FAC(CICSPROD)

--  Grant surrogate authority to CICS region user
TSS ADD(CICSP1) SURROGAT(PLTUSER)

--  Grant access to MQ File Control entries
TSS PER(PLTUSER) FCT(MQFCNFG) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFLOG)  ACC(ALL)
TSS PER(PLTUSER) FCT(MQFMON)  ACC(ALL)
TSS PER(PLTUSER) FCT(MQFERR)  ACC(ALL)
TSS PER(PLTUSER) FCT(MQFREOR) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFSSET) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFI001) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFI002) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFI003) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFO001) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFO002) ACC(ALL)
TSS PER(PLTUSER) FCT(MQFO003) ACC(ALL)

--  Grant authority to necessary transactions
TSS ADD(PLTUSER) TRANS(CICSPROD,INWL,IESO,IESN,MQIT,MQSM,MQTL)
```

Note that the CICS partition user must be a surrogate for the PLTPIUSR (that is, PLTUSER). Once again, to activate the surrogate user feature, you should include the following facilities matrix option:

```
FACILITY(CICSPROD=XUSER=YES)
```

If using BSM, then the SURROGAT class must be active and CICS started with XUSER=YES.

Also, take care that you do not grant NORESCHK to the PLTPIUSR. This is because resource checking for security switches will always result in success. Success indicates that the switch is present, and security features are deactivated. In other words, if the WebSphere MQ startup user, PLTPI or otherwise, has NORESCHK authority, WebSphere MQ resource security will be deactivated.

If you do not identify a SIT parameter for a PLTPIUSR, CICS TS uses the CICS default user. Although it may not be appropriate to authorize the CICS default user to WebSphere MQ resources, it is possible to use the default user for WebSphere MQ activation during CICS initialization.

Stopping WebSphere MQ

The MQADMUSR user has the authority to stop WebSphere MQ by running the MQST transaction (because it has authority to all WebSphere MQ transactions).

If you want to shut down WebSphere MQ via the PLTSD, you must ensure that the shutdown user is authorized to the appropriate resources relevant to the PLTSD phase (these may be other than WebSphere MQ resources).

The shutdown user is the user who issues the shutdown command, for example:

```
CEMT P SHUT
```

If this command is issued from the console, the console user must have authority similar to the PLTPIUSR user described earlier. Also, the shutdown user should have authority to execute the CEMT transaction.

Appendix I. WMQZVSE SOAP transport to z/VSE SOAP server

The WebSphere MQ transport for SOAP provides a JMS transport for SOAP.

A SOAP client using the Apache Axis 1.4 platform can send a web service request in a SOAP envelope to a z/VSE SOAP server using WebSphere MQ.

A SOAP listener program (MQPSOAPL) can be started as a service using start command MQSL. The service must specify the SOAP request queue and port number if the CICS running the web service does not have 8080 as its TCPIPService port number.

In the following example, the MQ SOAP listener is started when the queue manager is started and stopped when the queue manager is stopped.

```
2012/11/14        IBM WebSphere MQ for z/VSE Version 3.0.0           TSMQ300
07:14:21                    Service Record           DISPLAY         CIC1
MQWMSYS                                                              A001

Service name. . . . . :  MQSOAP.REQUESTS
Description . . . . . :  HANDLE SOAP REQUESTS PUT TO
                      :  "SOAP.REQUEST" QUEUE

Service type  . . . . :  C     (S=Server, C=Command)
Control . . . . . . . :  M     (M=Manual, Q=QMgr start/stop, S=QMgr start)
Start command . . . . :  MQSL
Start args  . . . . . :  Q=SOAP.REQUEST,SOAPPORT=8080
                      :
Stop command  . . . . :  MQSL
Stop args   . . . . . :  Q=SOAP.REQUEST
                      :

Service status. . . . :
Alteration date . . . :  2012-11-14
Alteration time . . . :  05:49:57

Requested record displayed.
PF2=Return PF3=Quit PF4=Read PF5=Add    PF6=Update   PF9=List
                             PF10=Start PF11=Stop    PF12=Delete
```

Figure 89. Add MQ SOAP service

The following sections describe how to create a web service from a CICS application and a sample client to use it. The sample is based on the WebSphere MQ transport for SOAP installation verification test described in the WebSphere MQ Web Services manual. This was run on the Windows platform.

Create web service from a CICS application

Refer to “How to use Web Services with z/VSE” on the IBM: z/VSE - Documentation - Documents - e-business web page.

The following example uses a web service created by using the CICS2WS toolkit available from the IBM: z/VSE - Downloads - Connectors page.

This was used to create a web service "MQCLservice" to link to the WebSphere MQ for z/VSE programmable interface to query the status of a queue. The following COBOL copybook is a simplified version of MQICMD.C and was used in the CICS2WS toolkit process.

```
01  MQI-COMMAND-LINE.
    02  MQI-CMD                PIC X(56).
    02  MQI-REPLY-RC           PIC 9(4) COMP.
    02  MQI-REPLY              PIC X(99).
    02  MQI-REPLY-STATUS       PIC X(16).
```

Also specified in CICS2WS were

```
Service name:      MQCLservice
Service URL:       http://n.n.n.n:8080/cics/CWBA/IESSOAPS

Proxy Name:        MQCLPRX
User Program Name: MQPCMD

Operation Name:    QUERY
```

A MQCLPRX.WSDL file and proxy assembler program were also generated. The WSDL file is generated for the client connecting to the web service using HTTP. In order to use WebSphere MQ to transport, the request had to be edited to change the address location from HTTP to JMS/SOAP transport bindings. This changes from

```
<wsdlsoap:address location="http://n.nnn.nnn.nn:8080/cics/CWBA/IESSOAPS"/>
```

to

```
<wsdlsoap:address location="jms:/queue?destination=SOAP.REQUEST@TEST.QM&
connectionFactory=(connectQueueManager(TEST.QM))&
initialContextFactory=com.ibm.mq.jms.Nojndi&
targetService=MQCLservice&replyDestination=SOAPJ.RESP"/>
```

The "connectQueueManager" is the queue manager name to which the SOAP client connects in order to put the SOAP request to the "destination" queue and gets the response from the "replyDestination queue". In the above example TEST.QM is WebSphere MQ running on the Windows platform. It has a remote queue SOAP.REQUEST which puts a SOAP request to the transmission queue. It has the remote queue name of "SOAP.REQUEST". This is the WebSphere MQ for z/VSE local queue which the SOAP listener is monitoring.

Channels have to be defined to send a SOAP request message to z/VSE and receive a SOAP response from z/VSE.

An alias queue manager should be defined with the reply queue manager name to allow the response to be returned

For example, a response will be put to RESP.XQ which is the transmission queue from the z/VSE system back to the Windows MQ reply destination queue.

```
2012/11/14        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
07:22:57               Queue Definition Record                      CIC1
MQWMQUE              Alias Queue Manager Definition                 A001



Object Name. . . . . . . . : TEST.QM
Description line 1 . . . . : ____________________________________
Description line 2 . . . . : ____________________________________

Alias Queue Manager Name . : TEST.QM
Transmission Queue Name. . : RESP.XQ









Record updated OK.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                      PF9=List PF12=Delete
```

Figure 90. Add Alias Queue Manager for MQ SOAP

Verifying the WebSphere MQ transport for SOAP

WebSphere MQ 7.1 was used on the Windows platform. The supplied sample assumes that the verification for Axis 1.4 has been run.

The ./tools/soap/samples is assumed to have been copied to the C:\TempSOAP directory. Refer to WebSphere MQ 7.1 documentation Verifying the WebSphere MQ transport for SOAP.

Running the WebSphere MQ for z/VSE sample java client using Axis

1. Catalog the MQ SOAP samples in MQSOAPEX.Z into a sublibrary

   ```
   * $$ JOB JNM=MQSOAP,CLASS=0
   * $$ LST CLASS=A,DISP=H
   // JOB MQSOAP
   // EXEC LIBR
   ACC S=mylib.mysublib
   * $$ SLI MEM=MQSOAPEX.Z,S=prd2.wmqzvse
   /*
   /&
   * $$ EOJ
   ```

   This should catalog the following:

   ```
   MQCLPRX.JCL
   MQCLPRX.A
   MQCLPRX.WSDL
   RUNMQCL.CMD
   RUNMQCL.JAVA
   MQCLCLNT.JAVA
   ```

2. Customize the MQCLPRX.JCL to assemble and link the MQCLPRX proxy program.
3. Define the MQCLPRX assembler program to CICS.
4. Create folders

```
C:\SOAP
C:\SOAP\JAVA
C:\SOAP\WSDL
```

5. From the mylib.mysublib download the

```
RUNMQCL.CMD    to  C:\SOAP
RUNMQCL.JAVA   to  C:\SOAP\JAVA
MQCLCLNT.JAVA  to  C:\SOAP\JAVA
MQCLPRX.WSDL   to  C:\SOAP\WSDL
```

   The java programs stored in the sublibrary use square brackets coded in code page 1047, that is "[" is `X'AB` and "]" is `X'DD`. After downloading to PC, check that they have been converted correctly. This may require you to change your mainframe code page for the download.

6. In the MQCLPRX.WSDL file, the "wsdlsoap:address location=" line has been split into many lines for storing in the sublibrary. You have to edit the file and delete the ### characters (including the CR/LF). You need to have `wsdlsoap:address location=' ... '/>` as one line.

   Customize to required object names the

```
destination=SOAP.REQUEST@TEST.QM
connectQueueManager(TEST.QM)
replyDestination=SOAPJ.RESP
```

7. The WebSphere MQ queue manager on Windows (TEST.QM in our example) has to have the remote queue defined (SOAP.REQUEST in our example), a response queue (SOAPJ.RESP in our example), a channel to send message to the WebSphere MQ for z/VSE and receiver channel to receive the SOAP response message.
8. In the z/VSE system, ensure CICS TS has TCPIPService open. If the port number is not 8080 then it has to be specified in the WebSphere MQ for z/VSE service definition for the SOAP listener using the `SOAPPORT=keyword` (refer example above).

   Any number of SOAP listener services may be defined, as long as each has a unique request queue.

9. Start the MQ SOAP listener service (if not already started).
10. Start the WebSphere MQ for Windows sender channel to WebSphere MQ for z/VSE.
11. In the Windows command prompt change to the `C:\SOAP` directory and run the `RUNMQCL` command.

    This should result in following being displayed:

```
MQCLCLNT: Response = 'MQM001000 QUEUE STATUS: IN=IDLE    ,OUT=IDLE
                      SYSTEM.LOG   '
```

    where the query was for status of SYSTEM.LOG queue.

    The following directories are assumed to exist following the running of the WebSphere MQ SOAP runivt.cmd to verify installation of WebSphere MQ SOAP on Windows.

```
C:\TempSOAP\soap\samples\java\clients
C:\TempSOAP\soap\samples\generated
C:\TempSOAP\soap\samples\classes\soap\clients
```

Appendix J. Publish/Subscribe

Publish/subscribe messaging allows you to decouple the provider of information, from the consumers of that information. The sending application and receiving application do not need to know anything about each other for the information to be sent and received. Before a point-to-point WebSphere MQ application can send a message to another application, it needs to know something about that application. For example, it needs to know the name of the queue to which to send the information, and might also specify a queue manager name.

WebSphere MQ publish/subscribe removes the need for your application to know anything about the target application. All the sending application has to do, is put a WebSphere MQ message, containing the information that it wants, and assign it a topic, that denotes the subject of the information, and let WebSphere MQ handle the distribution of that information.

Similarly, the target application does not have to know anything about the source of the information it receives.

Figure 91 shows the simplest publish/subscribe system. There is one publisher, queue manager, and one subscriber. A subscription request is sent from the subscriber to the queue manager, a publication is sent from the publisher to the queue manager, and the publication is then forwarded by the queue manager to the subscriber.

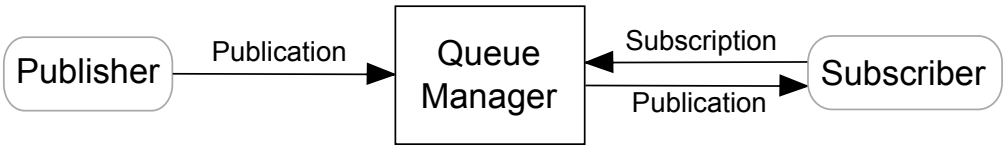


Figure 91. Simple publish/subscribe configuration

A typical publish/subscribe system has more than one publisher and more than one subscriber. An application can be both a publisher and a subscriber.

Refer to Publish/Subscribe entries in the IBM WebSphere MQ Information Center. WebSphere MQ for z/VSE supports a subset of what is described. Differences are described below.

Topics

A topic is the subject of the information that is published in a publish/subscribe message.

Messages in point-to-point systems are sent to a specific destination address. Messages in subject-based publish/subscribe systems are sent to subscribers based on the subject that describes the contents of the message.

The WebSphere MQ publish/subscribe system is a subject-based publish/subscribe system. A publisher creates a message, and publishes it with a topic string that best fits the subject of the publication. To receive publications, a subscriber creates a subscription with a pattern matching topic string to select publication topics. The

queue manager delivers publications to subscribers that have subscriptions that match the publication topic, and are authorized to receive the publications. Refer to "Topic strings" for description syntax of topic strings that identify the subject of a publication. Subscribers also create topic strings to select which topics to receive. The topic strings that subscribers create can contain the wildcard character “#” as the last node of a topic string.

In subject-based publish/subscribe, publishers, or administrators, are responsible for classifying subjects into topics. Typically subjects are organized hierarchically, into topic trees, using the “/” character to create subtopics in the topic string. See "Topic trees" for examples of topic trees. Topics are nodes in the topic tree. Topics can be leaf-nodes with no further subtopics, or intermediate nodes with subtopics.

In parallel with organizing subjects into a hierarchical topic tree, you can associate topics with administrative topic objects. You assign attributes to a topic by associating it with an administrative topic object. The association is made by naming the topic using the TOPICSTR attribute of the administrative topic object. If you do not explicitly associate an administrative topic object to a topic, the topic inherits the attributes of its closest ancestor in the topic tree that you have associated with an administrative topic object. If you have not defined any parent topics at all, it inherits from SYSTEM.BASE.TOPIC. Administrative topic objects are described in “Administrative topic objects” on page 1078.

Note: Even if you inherit all the attributes of a topic from SYSTEM.BASE.TOPIC, define a root topic for your topics that directly inherits from SYSTEM.BASE.TOPIC. For example, in the topic space of US states, USA/Alabama USA/Alaska, and so on, USA is the root topic. The main purpose of the root topic is to create discrete, non-overlapping topic spaces to avoid publications matching the wrong subscriptions. It also means you can change the attributes of your root topic to affect your whole topic space.

When you refer to a topic as a publisher or subscriber, you have a choice of supplying a topic string, referring to a topic object or you can do both, in which case the topic string you supply defines a subtopic of the topic object. The queue manager identifies the topic by appending the topic string to the topic string prefix named in the topic object, inserting an additional “/” in between the two topic strings, for example, topic string/object string. "Constructing topic names" describes this further. The resulting topic string is used to identify the topic and associate it with an administrative topic object.

The administrative topic object is not necessarily the same topic object as the topic object corresponding to the master topic.

Note: In z/VSE the maximum length of the resolved topic string is 256.

Topic strings

Label the information you publish as a topic using a topic string. Subscribe to groups of topics using either character- or topic-based wildcard topic strings.

A topic string is a character string that identifies the topic of a publish/subscribe message. You can use any characters you like when you construct a topic string.

Syntax

Two characters have special meaning in WebSphere MQ for z/VSE publish/subscribe.

A forward slash (/)
The topic level separator. Use the "/" character to structure the topic into a topic tree.

Avoid empty topic levels, "//" if you can. These correspond to nodes in the topic hierarchy with no topic string. A leading or trailing "/" in a topic string corresponds to a leading or trailing empty node and should be avoided too.

The hash sign (#)
Used in combination with "/" to construct a wildcard in subscriptions.

"#" is only supported as the final node, for example, `IBM/AUSTRALIA/#`, which matches any subscription with IBM and AUSTRALIA as the first two nodes of the topic tree. These topic strings would match for this example:

```
IBM/AUSTRALIA
IBM/AUSTRALIA/Perth
IBM/AUSTRALIA/Sydney/CBD
```

Using topic strings

When you refer to a topic as a publisher or subscriber, you have a choice of supplying a topic string, referring to a topic object or you can do both, in which case the topic string you supply defines a subtopic of the topic object. The queue manager identifies the topic by appending the topic string to the topic string prefix named in the topic object, inserting an additional "/" in between the two topic strings, for example, `topic string/object string`. The resulting topic string is used to identify the topic and associate it with an administrative topic object

Topic trees

Each topic that you define is an element, or node, in the topic tree. The topic tree can either be empty to start with or contain topics that have been defined previously using the admin panels, MQSC or PCF commands. You can define a new topic either by using the create topic commands or by specifying the topic for the first time in a publication or subscription.

Although you can use any character string to define a topic's topic string, it is advisable to choose a topic string that fits into a hierarchical tree structure. Thoughtful design of topic stings and topic trees can help you with the following operations:
- Subscribing to multiple topics
- Establishing security policies

Although you can construct a topic tree as a flat, linear structure, it is better to build a topic tree in a hierarchical structure with one or more root topics. For more information about security planning and topics, see "Publish/subscribe security".

Here is an example of a topic tree with one root topic.

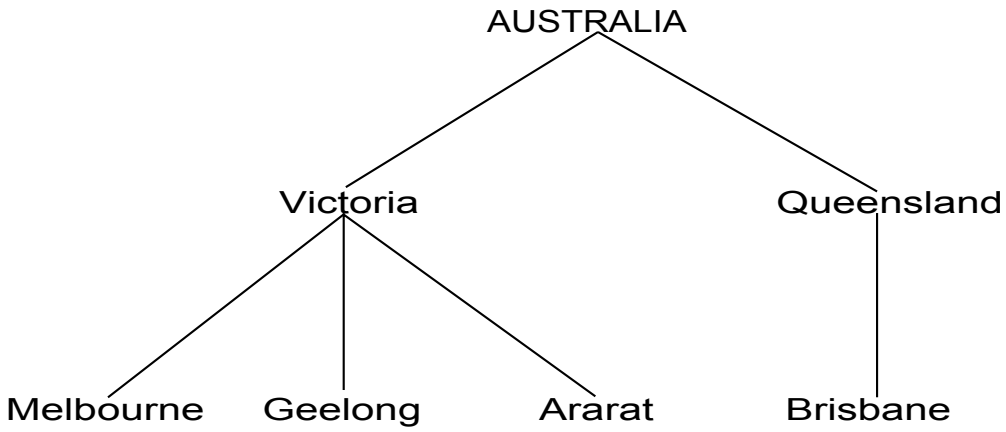


Figure 92. Example of a topic tree

Each character string in the figure represents a node in the topic tree. A complete topic string is created by aggregating nodes from one or more levels in the topic tree. Levels are separated by the "/" character. The format of a fully specified topic string is `root/level2/level3`.

The valid topics in this topic tree are:

```
"AUSTRALIA"
"AUSTRALIA/Victoria"
"AUSTRALIA/Queensland"
"AUSTRALIA/Victoria/Melbourne"
"AUSTRALIA/Victoria/Geelong"
"AUSTRALIA/Victoria/Ararat"
"AUSTRALIA/Queensland/Brisbane"
```

When you design topic strings and topic trees, remember that the queue manager does not interpret, or attempt to derive meaning from, the topic string itself. It simply uses the topic string to send selected messages to subscribers of that topic.

The following principles apply to the construction and content of a topic tree:

- The number of levels in a topic tree has to fit in the topic string z/VSE limit of 256 bytes.
- The length of the name of a level in a topic tree has to fit in the topic string z/VSE limit of 256 bytes.
- There can be any number of "root" nodes; that is, there can be any number of topic trees.

Administrative topic objects

An administrative topic object is a WebSphere MQ object that allows you to assign specific, non-default attributes to topics. A topic is defined by an application publishing or subscribing to a particular topic string. A topic string can specify a hierarchy of topics by separating them with a forward slash character (/). This can be visualized by a topic tree. For example, if an application publishes to the topic strings "Sport/American Football" and "Sport/Soccer", a topic tree is created that has a parent node Sport with two children, "American Football", and "Soccer".

Topics inherit their attributes from the first parent administrative node found in their topic tree. If there are no administrative topic nodes in a particular topic tree, then all topics inherit their attributes from the base topic object, "SYSTEM.BASE.TOPIC".

You can create an administrative topic object at any node in a topic tree by specifying that node's topic string in the TOPICSTR attribute of the administrative topic object. You can also define other attributes for the administrative topic node. For more information about these attributes, see the admin panel, MQSC or PCF commands. Each administrative topic object, by default, inherits its attributes from its closest parent administrative topic node.

Administrative topic objects can also be used to hide the full topic tree from application developers. If an administrative topic object named FOOTBALL.US is created for the topic "Sport/American Football", an application can publish or subscribe to the object named FOOTBALL.US instead of the string "Sport/American Football" with the same result.

If you enter a # character within a topic string on a topic object, the character is treated as a normal character within the string, and is considered to be part of the topic string associated with an administrative topic object.

Administrative panels

The Configuration Main Menu includes the Topic and Subscription functions.

```
2012/11/26      IBM WebSphere MQ for z/VSE Version 3.0.0            TSMQ300
12:59:55           *** Configuration Main Menu ***                  CIC1
MQWMCFG                                                             A001
                            SYSTEM IS ACTIVE

                Maintenance/Display Options
                          1 / 8     Global System Definitions
                          2 / 9     Queue Definitions
                          3 / 10    Channel Definitions
                          4 / 11    Code Page Definitions
                          5 / 12    Namelist Definitions
                          6 / 13    Topic Definitions
                          7 / 14    Subscription Definitions

                                 Option: __




Please enter one of the options listed.
          5655-U97 Copyright IBM Corp. 2008. All rights reserved.
Enter=Process  PF2=Return  PF3=Exit
```

Figure 93. Configuration Main Menu

To maintain a TOPIC select option 6, to display TOPIC select option 13. If you wish to secure the transactions for TOPICs, MQM1 is used for maintenance and MQD1 for display only.

To maintain an administrative SUBSCRIPTION select option 7, to display select option 14. If you wish to secure the transactions for SUBSCRIPTIONs, MQM2 is used for maintenance and MQD2 for display only.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:01:50                 Topic Name Definition                      CIC1
MQWMTOP                                                             A001
                         Queue manager active


       Topic Name . . . . : ____________________________________________




Key topic name or PF9 to list topics

PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add
                                      PF9=List PF12=Delete
```

Figure 94. Topic Name Definition

Key the topic name to update/add or PF9 to list topics. The topic name has to be standard MQ object name consisting of alphanumerics (upper and lower case) and characters . / _ %

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:03:57                    Topic Name List                         CIC1
MQWMTOP                                                             A001
S Topic name
_ AIRLINES
_ CARS
_ FRUIT
_ SYSTEM.BASE.TOPIC
_ SYSTEM.DEFAULT.TOPIC
_
_
_
_
_
_
_
_
_
_
_
List scrolled.
PF2=Return  PF3=Quit     PF4/Enter=Read
            PF7=Backward PF8=Forward
```

Figure 95. Topic Name List

Key any character to select a listed topic and press Enter.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:05:58                   Topic Name Definition                    CIC1
MQWMTOP                                                             A001

Topic name . . . . . . . . : CARS
Description line 1 . . . . : Cars topic root
Description line 2 . . . . :

Topic String . . . . . . . : Cars




Inhibit Publication. . . . : P    (P=ASPARENT, N=NO, Y=YES)
Inhibit Subscription . . . : P    (P=ASPARENT, N=NO, Y=YES)
Durable  . . . . . . . . . : P    (P=ASPARENT, N=NO, Y=YES)
Model durable queue. . . :
Model non-durable queue. :

Message Delivery . . . . . : P    P=ASPARENT,V=ALL AVAIL,D=ALL DUR,A=ALL)

Topic record displayed.
PF2=Return  PF3=Quit  PF4/Enter=Read
                                    PF9=List
```

Figure 96. Maintain Topic Name

The Topic String field specifies the topic string to be associated with this topic.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:42:34                   Sub Name Definition                      CIC1
MQWMSUB                                                             A001
                         Queue manager active


       Sub Name . . . . . : ____________________________________________




Key SubName or PF9 to list SubNames

PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add
                                    PF9=List PF12=Delete
```

Figure 97. Subscription Name Definition

Key the subscription name to update or add, or press PF9 to list topics.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:46:42                     Sub Name List                          CIC1
MQWMSUB                                                             A001
S Topic name
_ CHERRY
_ SYSTEM.DEFAULT.SUB
_
_
_
_
_
_
_
_
_
_
_
_
_
_

SubName(s) listed - select a topic
PF2=Return  PF3=Quit     PF4/Enter=Read
            PF7=Backward PF8=Forward
```

Figure 98. Subscription Name List

Key any character to select listed subscription and press Enter.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:53:00                   Sub Name - General                       CIC1
MQWMSUB                                                             A001

Subscription Name. . . . : BANANA

TOPIC
Topic Name . . . . . . . : PRICE

Topic String . . . . . . : Banana
_____________________________________________________________________
_____________________________________________________________________
_____________________________________

DESTINATION
Destination Class. . . . : m    M=Managed  P=Provided
Destination Queue Manager: _____________________________________________
Destination Queue. . . . : _____________________________________________

Properties . . . . . . . : N

SubName record being added - press PF5 to complete ADD
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add        PF9=List
                                      PF10=Extended
```

Figure 99. Maintain Subscription General

Adding subscription "BANANA" the resolved topic string is "Price/Banana". The PRICE topic string is concatenated with “/” and the entered topic string of “Banana”.

```
2012/11/26        IBM WebSphere MQ for z/VSE Version 3.0.0          TSMQ300
13:55:05                   Sub Name - Extended                      CIC1
MQWMSUB                                                             A001

Subscription Name. . . . : BANANA

Variable user ID . . . . : A    (A = Any, F=Fixed)

User . . . . . . . . . . : ____________

Application Identity . . : ____________________________

Expiry . . . . . . . . . : 99999999

Request Only . . . . . . : A    A=All or O=On Request




SubName record being added - press PF5 to complete ADD
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add        PF9=List
                                      PF10=General
```

Figure 100. Maintain Subscription Extended

Post PTF application

- Customize the durable and non-durable model queues for message size and number of messages. Sample definitions are provided in MQJINSG.Z. The default model names are SYSTEM.DURABLE.MODEL.QUEUE with file name MQFDUR and SYSTEM.NDURABLE.MODEL.QUEUE with file name MQFNDUR.
- For retained publications, queue name SYSTEM.RETAINED.PUB.QUEUE with file name MQFPUBR has to be defined. This is not configurable. Customize the sample in MQJINSG.Z for maximum message size.
- Regardless of the durable and non-durable model queue names specified, the dynamic queue name used is always SYSTEM.MANAGED.DURABLE.* or SYSTEM.MANAGED.NDURABLE.*.
- CSD definitions have to be updated. Refer MQJCSD24.Z.
- Customize MQJSETUP.Z to load configuration file updates to the work file and then run MQSU before starting MQ.
- If using CICS Web Support in place of the admin panels then you need to load the updated html files to the DFHDOC sublibrary.

Notes

- The subscription name is limited to 48 characters.
- The resolved topic string is limited to 256 characters.
- The messages are published outside the unit of work of the publisher, that is, as if using MQPMO_NO_SYNCPOINT.
- If using MQ Explorer then some options are not supported. For example only QMGR is supported for Publication scope or Subscription scope. Generally options not supported (if selected) are ignored.
- When using the MQSC utility program, names and topic strings should be enclosed in apostrophes if the case is to be maintained. For example, `TOPICSTR(Price/Fruit/Apples)` results in the topic string “PRICE/FRUIT/APPLES”.

To maintain the lowercase characters use `TOPICSTR('Price/Fruit/Apples')`.

Security

To use BSM to secure publish or subscribe you need to have one of the following ptfs applied:

z/VSE 4.3
UD53859

z/VSE 5.1
UD53860

Table 78 shows, for each WebSphere MQ PCF command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 79 shows, for each WebSphere MQ MQSC command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 78. PCF commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level MQCMDS	Resource profile for MQADMIN	Access level for MQADMIN
change SUB	hlq.ALTER.SUB	ALTER	No check	-
change TOPIC	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
copy SUB	hlq.DEFINE.SUB	ALTER	No check	-
copy TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
create SUB	hlq.DEFINE.SUB	ALTER	No check	-
create TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
delete SUB	hlq.DELETE.SUB	ALTER	No check	-
delete TOPIC	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
inquire PUBSUB	hlq.DISPLAY.PUBSUB	READ	No check	-
inquire SBSTATUS	hlq.DISPLAY.SBSTATUS	READ	No check	-
inquire SUB	hlq.DISPLAY.SUB	READ	No check	-
inquire TOPIC	hlq.DISPLAY.TOPIC	READ	No check	-
inquire TOPIC names	hlq.DISPLAY.TOPIC	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-

Table 79. MQSC commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level MQCMDS	Resource profile for MQADMIN	Access level for MQADMIN
change SUB	hlq.ALTER.SUB	ALTER	No check	-
change TOPIC	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
copy SUB	hlq.DEFINE.SUB	ALTER	No check	-
copy TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
create SUB	hlq.DEFINE.SUB	ALTER	No check	-
create TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER

Table 79. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level MQCMDS	Resource profile for MQADMIN	Access level for MQADMIN
delete SUB	hlq.DELETE.SUB	ALTER	No check	-
delete TOPIC	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
inquire PUBSUB	hlq.DISPLAY.PUBSUB	READ	No check	-
inquire SBSTATUS	hlq.DISPLAY.SBSTATUS	READ	No check	-
inquire SUB	hlq.DISPLAY.SUB	READ	No check	-
inquire TOPIC	hlq.DISPLAY.TOPIC	READ	No check	-
inquire TOPIC names	hlq.DISPLAY.TOPIC	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-

Grant access to a user to publish to a topic

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.

If no topic object is provided then the queue manager goes up the topic tree to find an administrative topic. If none is found then the base topic SYSTEM.BASE.TOPIC is used.

For example, on the queue manager with Subsystem id MQV1, allow user MQU1 to publish on the topic "Apples":

```
ADD    MQADMIN 'MQV1.TOPIC.Apples'       UACC(NONE)
PERMIT MXTOPIC 'MQV1.PUBLISH.Apples'     ID(MQU1) ACCESS(UPDATE)
```

Grant access for subscribe

To subscribe to a topic, you need access to both the topic you are trying to subscribe to, and the target queue for the publications. When you issue an MQSUB request, the following security checks take place:

- Whether you have the appropriate level of access to subscribe to that topic, and also that the target queue (if specified) is opened for output.
- Whether you have the appropriate level of access to that target queue.

Table 80. Access level required for topic security to subscribe

Action	Access level required
MQSUB to topic	hlq.SUBSCRIBE.topicname profile in MXTOPIC class
MQSO_CREATE	ALTER
MQSO_RESUME	READ

For example:

```
ADD    MXTOPIC 'MQV1.SUBSCRIBE.Apples'   UACC(NONE)
PERMIT MXTOPIC 'MQV1.SUBSCRIBE.Apples'   ID(MQU1) ACCESS(UPDATE)
```

Table 81. Access level required to profiles for topic security for closure of a subscribe operation

Action	Access level required
MQCLOSE sub	hlq.SUBSCRIBE.topicname profile in MXTOPIC class
MQCO_REMOVE_SUB	ALTER

Considerations for managed queues for subscriptions

A security check is carried out to see if you are allowed to subscribe to the topic. However, no security checks are carried out when the managed queue is created, or to determine if you have access to put messages to this destination queue. You cannot close delete a managed queue. The default model queues used are:SYSTEM.DURABLE.MODEL.QUEUE and SYSTEM.NDURABLE.MODEL.QUEUE.

The managed queues created from these model queues are of the form SYSTEM.MANAGED.DURABLE.A346EF00367849A0 and SYSTEM.MANAGED.NDURABLE.A346EF0036785EA0 where the last qualifier is unpredictable.

Do not give any user access to these queues. The queues can be protected using generic profiles of the form SYSTEM.MANAGED.DURABLE.* and SYSTEM.MANAGED.NDURABLE.* with no authorities granted.

Messages can be retrieved from these queues using the handle returned on the MQSUB request.

Appendix K. 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 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.

Use of channel authentication records to control inbound channels is enabled using the ALTER QMGR CHLAUTH(ENABLED) switch or using the admin panel Global System Definition.

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 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.

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.

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.

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.

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 a trailing wildcard (“*”). 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.

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.

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.

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.

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, 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 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 or queue manager name, 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.
- 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 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 dot part separator . 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.
- For a queue manager name:
  - The most specific match is a name without wildcard, for example PTHVSEC
  - A name with trailing asterisk is not a specifc as one without, for example, “PTHVSEC” is more specific than “PTHVSE*”.
  - The most generic match is a single asterisk (*), which matches all manager names.

WebSphere MQ Explorer

When using MQ Explorer connected to WebSphere MQ for z/VSE to create a new channel authentication record then you cannot use the identity match option "SSL/TLS subject's Distinguished Name". This is not supported in WebSphere MQ for z/VSE.

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information 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 information. The furnishing of this information does not give 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:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

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 United Kingdom Laboratories, Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

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 Programming License Agreement, or any equivalent agreement between us.

Copyright license

Trademarks

Glossary of terms and abbreviations

This glossary defines WebSphere MQ terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

adapter
An interface between WebSphere MQ for z/OS and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access WebSphere MQ services.

address space
The area of virtual storage available for a particular job.

address space identifier (ASID)
A unique, system-assigned identifier for an address space.

Adopt MCA
An WebSphere MQ feature that allows an MCA instance to "adopt" the function of an existing MCA when it is deemed to have stalled.

alert A message sent to a management services focal point in a network to identify a problem or an impending problem.

alert monitor
In WebSphere MQ for z/OS, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to WebSphere MQ for z/OS.

alias queue object
An WebSphere MQ object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

ally An MVS address space that is connected to WebSphere MQ for z/OS.

APAR Authorized program analysis report.

API exit
A user-written program called by the queue manager before and after MQI calls are processed.

application environment
The software facilities that are accessible by an application program. On the z/OS platform, CICS and IMS are examples of application environments.

application queue
A queue used by an application.

ASID Address space identifier.

asynchronous messaging
A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute
One of a set of properties that defines the characteristics of an WebSphere MQ object.

authorization checks
Security checks that are performed when a user tries to open an WebSphere MQ object.

authorized program analysis report (APAR)
A report of a problem caused by a suspected defect in a current, unaltered release of a program.

auto-definition
See *channel auto-definition*.

auto-definition event
See*channel auto-definition event*.

auto-definition exit
See*channel auto-definition exit*.

B

backout
An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

basic mapping support (BMS)
An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

batch auto-start
A queue manager parameter used to indicate whether or not the WebSphere MQ batch interface should be started automatically during system initialization.

batch identifier
An XPCC identifier used to uniquely identify a queue manager to batch applications.

batch interval
An interval in milliseconds for a batch of messages to stay active before the queue manager deems the batch is complete and commits the transmitted messages.

BMS Basic mapping support.

browse
In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor
In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

Bullet-proof
A feature of WebSphere MQ channels that allows for a channel to wait no longer than a configurable period of time to receive data.

C

call back
In WebSphere MQ, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCSID
Coded character set identifier.

chained exit
A user-written exit program called in a chain of exit programs.

channel auto-definition
An WebSphere MQ feature that facilitates the automatic creation of message channel definitions as they are needed.

channel auto-definition event
An event message generated when a channel auto-definition occurs or fails.

channel auto-definition exit
A user-written exit that controls the automatic creation of message channel definitions.

CDF Channel definition file.

channel
See *message channel*.

channel definition file (CDF)
In WebSphere MQ, a file containing communication channel definitions that associate transmission queues with communication links.

channel event
An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint
A time when significant information is written on the log. Contrast with *syncpoint*.

CI Control interval.

class For security, a class associates a group of resources. WebSphere MQ uses the security classes MQADMIN, MQCONN and MQQUEUE.

client A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *WebSphere MQ client*.

client application
An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client bridge
A component of the WebSphere MQ

client, unique to z/VSE, that allows applications running in a non-LE environment to use the client MQI.

client connection channel type
The type of MQI channel definition associated with an WebSphere MQ client. See also *server connection channel type*.

COA Confirm-on-arrival. In reply queue processing, a reply message can be generated when a message is initially put to a queue by using the COA report option in the message descriptor of an object message.

COD Confirm-on-delivery. In reply queue processing, a reply message can be generated when a message is initially read from a queue by using the COD report option in the message descriptor of an object message.

coded character set identifier (CCSID)
The name of a coded set of characters and their code point assignments.

command
In WebSphere MQ, an instruction that can be carried out by the queue manager.

command processor
An WebSphere MQ program responsible for processing PCF messages. The command processor validates and executes PCF commands, and generates response messages to the issuer.

command resource security
Security pertaining to WebSphere MQ commands issued against WebSphere MQ resources.

command server
An WebSphere MQ program responsible for processing the system command queue. The command server reads PCF message from the command queue and starts an instance of the WebSphere MQ command processor to process the PCF message.

command server auto-start
A queue manager parameter used to indicate whether or not theWebSphere MQ command server should be started automatically during system initialization.

commit
An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code
A return code indicating how an MQI call has ended.

connect
To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN call, or automatically by the MQOPEN call.

connection handle
The identifier or token by which a program accesses the queue manager to which it is connected.

context
Information about the origin of a message.

control interval (CI)
A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

CWS CICS Web Support. A feature of CICS TS that allows CICS transactions to be run from a web browser.

D

DAE Dump analysis and elimination.

DBCS In data conversion, a Double Byte Character Set.

DCT In CICS, the Destination Control Table.

dead-letter queue (DLQ)
A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

default object
A definition of an object (for example, a queue) with all attributes defined.

disconnection interval
An interval in seconds that a channel will remain active once the transmission queue is empty.

distributed application
In message queuing, a set of application programs that can each be connected to a

different queue manager, but that collectively constitute a single application.

distribution list
A list of queues to which a message can be put using a single MQPUT or MQPUT1 statement

distributed queue management (DQM)
In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ Dead-letter queue.

DQM Distributed queue management.

dump analysis and elimination (DAE)
A z/OS service that enables an installation to suppress SVC dumps and ABEND SYSUDUMP dumps that are not needed because they duplicate previously written dumps.

dynamic queue
A queue created dynamically by an application program using MQOPEN to open a model queue. Dynamic queues can be permanent or temporary.

E

environment
See *application environment*.

ESM External security manager.

event See *instrumentation event*.

event data
In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also event header.

event header
In an event message, the part of the message data that identifies the event type of the reason code for the event.

event log
See *event queue*.

event message
Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of WebSphere MQ systems.

event queue
The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

exit A program called at defined places in the processing carried out by the queue manager or MCA programs.

Explorer
See *WebSphere MQ Explorer*.

external security manager (ESM)
A security product that is invoked by the z/OS System Authorization Facility. RACF® is an example of an ESM.

F

FCT In CICS, the File Control Table.

FIFO First-in-first-out.

first-in-first-out (FIFO)
A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

forced shutdown
A type of shutdown of the CICS adapter where the adapter immediately disconnects from WebSphere MQ for z/OS, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

FRR Functional recovery routine.

functional recovery routine (FRR)
A z/OS recovery/termination manager facility that enables a recovery routine to gain control in the event of a program interrupt.

G

get In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

group See *message group*.

H

handle
See *connection handle* and *object handle*.

I

immediate shutdown
In WebSphere MQ, a shutdown of a

queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

initiation queue
A local queue on which the queue manager puts trigger messages.

input/output parameter
A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

instrumentation event
In WebSphere MQ, an event is a logical combination of conditions that is detected by a queue manager or channel instance.

IP Address
Internet Protocol address. Usually a four-part dotted decimal value that uniquely identifies a remote host, for example, 1.20.33.444.

ISO International Standards Organization. In data conversion, ISO code pages are those that conform to ISO definitions.

L

listener
A communications program that runs while WebSphere MQ is active. The Listener program waits for connection requests from Sender MCAs or from client programs. For WebSphere MQ for z/VSE V3.1, the Listener exclusively waits for TCP/IP connection requests and starts the Receiver MCA.

local definition
An WebSphere MQ object belonging to a local queue manager.

local definition of a remote queue
An WebSphere MQ object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

local queue
A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager
The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

log In WebSphere MQ, a file recording the work done by queue managers while they receive, transmit, and deliver messages.

logical message
A single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message.

M

machine check interrupt
An interruption that occurs as a result of an equipment malfunction or error. A machine check interrupt can be either hardware recoverable, software recoverable, or non-recoverable.

MCA Message channel agent.

message
In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. In system programming, information intended for the terminal operator or system administrator.

message channel
In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender and a receiver) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA)
A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

message descriptor
Control information describing the message format and presentation that is

carried as part of an WebSphere MQ message. The format of the message descriptor is defined by the MQMD structure.

message exit
An exit program called during channel operation following the retrieval of a message from a queue and prior to a message being placed on a queue.

message expiry
Message attribute identifying a period of time expressed in tenths of a second. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

message group
A group of logical messages. Logical grouping of messages allows applications to group messages that are similar and to ensure the sequence of the messages.

message priority
In WebSphere MQ, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue
Synonym for *queue*.

message queue interface (MQI)
The programming interface provided by the WebSphere MQ queue managers. This programming interface allows application programs to access message queuing services.

message queuing
A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message segment
One of a number of segments of a message that is too large either for the application or for the queue manager to handle.

messaging
See *synchronous messaging* and *asynchronous messaging*.

model queue
A template of a queue definition, used to create a dynamic queue. When a model queue is opened using MQOPEN, a dynamic queue is created with the attributes of the model queue.

MQI Message queue interface.

MQI channel
Connects an WebSphere MQ client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQMD
WebSphere MQ Message Descriptor. The MQMD is a data structure that is prefixed to all WebSphere MQ messages.

MQSC
WebSphere MQ Command. MQSC commands are verb-based text commands that manipulate or display WebSphere MQ objects.

WebSphere MQ
A family of IBM licensed programs that provides message queuing services.

WebSphere MQ client
Part of an WebSphere MQ product that can be installed on a system without installing the full queue manager. The WebSphere MQ client accepts MQI calls from applications and communicates with a queue manager on a server system.

N

namelist
An WebSphere MQ object that contains a list of names, such as queue names.

null character
The character that is represented by X'00'.

O

object In WebSphere MQ, an object is a queue manager, a queue, a process definition, a channel, a namelist (z/OS only), or a storage class (z/OS only).

object descriptor
A data structure that identifies a particular WebSphere MQ object. Included in the descriptor are the name of the object and the object type.

object handle
The identifier or token by which a program accesses the WebSphere MQ object with which it is working.

output parameter
A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

page set
A VSAM data set used when WebSphere MQ for z/OS moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

PCF Programmable Command Format. A data message containing an WebSphere MQ command and associated parameters. PCF messages are written to the system command queue.

PCT In CICS, the Program Control Table.

pending event
An unscheduled event that occurs as a result of a connect request from a CICS adapter.

performance event
A category of event indicating that a limit condition has occurred.

permanent dynamic queue
A dynamic queue that retains messages when the queue manager is stopped and restarted, and remains until it is explicitly deleted.

persistent message
A message that survives a restart of the queue manager.

physical message
The smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call.

ping In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

PKI Public Key Infrastructure. The PKI infrastructure includes X.509 digital certificates used by SSL services.

platform
In WebSphere MQ, the operating system under which a queue manager is running.

preemptive shutdown
In WebSphere MQ, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

port A unique communications identifier used by TCP/IP programs to establish a conversation with a specific application. The target application binds a TCP/IP socket to the unique port number and then waits for connection requests for the port from remote hosts.

PPT In CICS, the Processing Program Table.

program temporary fix (PTF)
A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF Program temporary fix.

Q

queue An WebSphere MQ object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager
A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. An WebSphere MQ object that defines the attributes of a particular queue manager.

queue manager event
An event that indicates:
- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queuing
See *message queuing*.

quiesced shutdown
In WebSphere MQ, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. A type of shutdown of the CICS adapter where the adapter disconnects from WebSphere MQ, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing
In WebSphere MQ, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

reason code
A return code that describes the reason for the failure or partial success of an MQI call.

receive exit
An exit program called immediately following the receipt of data over an active channel.

receiver channel
In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

remote queue
A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager
To a program, a queue manager that is not the one to which the program is connected.

remote queue object
See *local definition of a remote queue*.

remote queuing
In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message
A type of message used for replies to request messages.

reply-to queue
The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message
A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason.

requester channel
In message queuing, a receiver-type channel that can activate a remote sender or server channel. Following channel activation, the requester channel acts as a receiver channel. See also *server channel*.

request message
A type of message used to request a reply from another program.

resolution path
The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource
Any facility of the computing system or operating system required by a job or task.

resource manager
An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. WebSphere MQ, CICS, and IMS are resource managers.

return codes
The collective name for completion codes and reason codes.

rollback
Synonym for *back out*.

S

SAF System Authorization Facility. SAF is an interface between the z/VSE operating system and external security managers. The SAF interface is used for security purposes.

SBCS In data conversion, a Single Byte Character Set.

security exit
An exit program called during the establishment of a channel.

segment
See *message segment*.

segmentation
A function where messages can be broken down into segments, or reassembled into a complete message automatically by the queue manager.

sender channel
In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

Send exit
An exit program called prior to the transmission of data over an active channel.

sequential number wrap value
In WebSphere MQ, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server (1) In WebSphere MQ, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel
In message queuing, a channel that acts as a sender-type channel after it has been activated by a remote requester channel. Following channel activation, the server channel acts as a sender channel. See also *requester channel*.

server connection channel type
The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

session ID
In WebSphere MQ for z/OS, the CICS-unique identifier that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

shutdown
See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

single-phase commit
A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

socket A communications handle used by TCP/IP programs to send data to, and receive data from, a remote host.

SSID Subsystem Identifier. An SSID is usually synonymous with an WebSphere MQ queue manager name.

SSL Secure Sockets Layer. A integrated feature of the TCP/IP product that provides a set of services to secure e-business transactions, including data encryptions and X.509 certificate exchange.

subsystem
In z/OS, a group of modules that provides function that is dependent on z/OS. For example, WebSphere MQ for z/OS is a z/OS subsystem.

supportpac
An IBM delivery package for providing licensed product features. Available supportpacs can generally be downloaded from the IBM website. Each supportpac has its own license agreement.

synchronous messaging
A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint
An intermediate or end point during processing of a transaction at which the transaction's protected resources are

consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

system command queue
The system command queue is a communication parameter of the global system definition and identifies the target queue for PCF command messages.

system reply queue
The system reply queue is a communication parameter of the global system definition and identifies the target queue for MQSC response messages.

system initialization table (SIT)
A table containing parameters used by CICS on start up.

T

target library high-level qualifier (thlqual)
High-level qualifier for z/OS target data set names.

TCP/IP
Transmission Control Protocol, Internet Protocol. TCP/IP is a family of communications protocols.

temporary dynamic queue
A dynamic queue that does not retain messages when the queue manager is stopped and restarted, and is deleted when the application that created it closes the queue or terminates.

thlqual
Target library high-level qualifier.

thread In WebSphere MQ, the lowest level of parallel execution available on an operating system platform.

trace In WebSphere MQ, a facility for recording WebSphere MQ activity. The destinations for trace entries can include GTF and the system management facility (SMF).

tranid See *transaction identifier*.

transaction identifier
In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

transmission queue
A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

triggering
In WebSphere MQ, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message
A message containing information about the program that a trigger monitor is to start.

trigger monitor
A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

two-phase commit
A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

undelivered-message queue
See *dead-letter queue*.

Unicode
Codepage UCS-2 is the Universal Multiple-Octet Coded Character Set defined by ISO/IEC 10646-1:1993(EE).

unit of recovery
A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work
A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

utility In WebSphere MQ, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the WebSphere MQ commands. Some utilities invoke more than one function.

W

WebSphere MQ Explorer
A GUI interface available on Windows and Linux (x86) that facilitates administration and monitoring of remote queue managers.

X

X.509 X.509 is the standard used for the generation and interpretation of PKI certificates.

Bibliography

This bibliography describes the documentation available for all current WebSphere MQ products.

WebSphere MQ cross-platform publications

For each WebSphere MQ cross-platform publication you will find the title and order number, followed by a brief description of the content of the publication and the intended audience, to help you decide whether you need that document.

These cross-platform publications apply to these WebSphere MQ products, unless otherwise stated in the book:

- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ for i5/OS
- WebSphere MQ for Linux for Intel
- WebSphere MQ for Linux for zSeries
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- WebSphere MQ for z/OS

An Introduction to Messaging and Queuing

An Introduction to Messaging and Queuing, GC33-0805, defines the problems solved by WebSphere MQ products, explains how messaging and queuing works, examines its chief benefits, provides a small amount of technical detail about messages and message queues, and examines some typical real world uses.

This book is for anyone new to the subject of messaging and queuing.

WebSphere MQ Application Programming Guide

WebSphere MQ Application Programming Guide, SC34-6939, introduces the concepts of messages and queues, and shows you in detail how to design and write applications that use the services that WebSphere MQ provides.

This book is for the designers of applications that will use message queuing techniques, and for programmers who have to implement these designs. To write message queuing applications using WebSphere MQ, you need to know how to write programs in at least one of the programming languages that WebSphere MQ supports. However, to understand this book, you do not need to have written message queuing programs before.

WebSphere MQ Application Programming Reference

WebSphere MQ Application Programming Reference, SC34-6940, introduces the concepts of messages and queues, and gives a full description of the WebSphere MQ programming interface, including data types, function calls, attributes, return codes, and constants.

This book is for the designers of applications that will use message queuing techniques, and for programmers who have to implement these designs. To write message queuing applications using WebSphere MQ, you need to know how to

write programs in at least one of the programming languages that WebSphere MQ supports. However, to understand this book, you do not need to have written message queuing programs before.

WebSphere MQ Clients

WebSphere MQ Clients, GC34-6934, describes the WebSphere MQ client/server environment. It describes how to install WebSphere MQ clients on different platforms, how to configure communications for different protocols, and how to define WebSphere MQ channels for client and server connections, in step-by-step instructions, complete with examples. It goes on to show you how to use WebSphere MQ applications in a WebSphere MQ client/server environment, including how WebSphere MQ applications connect to a queue manager.

This book is for anyone who installs and configures WebSphere MQ clients and WebSphere MQ servers, for system administrators, and for application programmers who write programs that use the Message Queue Interface (MQI). To understand this book, you should have experience in installing and configuring the system you use for the server, experience with any client platforms that you will be using, an understanding of the purpose of the Message Queue Interface (MQI), and experience of WebSphere MQ programs in general, or familiarity with the content of the other WebSphere MQ publications.

WebSphere MQ Constants

WebSphere MQ Constants, SC34-6951, provides a full description of the constants used by WebSphere MQ.

This book is for the designers of applications that use message queuing techniques, and for programmers who have to implement these designs. To write message queuing applications using WebSphere MQ, you need to know how to write programs in one of the supported programming languages. To understand this book, you do not need to have written message queuing programs before.

Monitoring WebSphere MQ

Monitoring WebSphere MQ, SC34-6937, describes how to use instrumentation events, activity reports, and the trace route facility, in a network of connected systems that use WebSphere MQ products in different operating system environments.

This book is for system programmers who write programs to monitor and administer WebSphere MQ products. To understand it, you need experience in writing systems management applications, an understanding of the Message Queue Interface (MQI), and experience of WebSphere MQ programs in general, or familiarity with the content of the other books in the WebSphere MQ library.

WebSphere MQ Intercommunication

WebSphere MQ Intercommunication, SC34-6931, describes intercommunication between WebSphere MQ products. It introduces the concepts of intercommunication (transmission queues, message channel agent programs, and communication links) that are brought together to form message channels. It describes how geographically-separated queue managers are linked together by message channels to form a network of queue managers. It discusses the distributed-queuing management (DQM) facility of WebSphere MQ, which provides the services that enable applications to communicate using queue managers.

This book is for anyone who needs to use WebSphere MQ intercommunication facilities including:

- Network planners responsible for designing the overall queue manager network.
- Local channel planners responsible for implementing the network plan on one node.
- Application programmers responsible for designing applications that include processes, queues, and channels.
- Systems administrators responsible for monitoring the local system, controlling exception situations, and implementing some of the planning details.
- System programmers responsible for designing and programming user exits.

To use and control DQM you need a good knowledge of WebSphere MQ in general. You also need to understand the WebSphere MQ products for the platforms you will be using, and the communications protocols used on those platforms.

WebSphere MQ Messages

WebSphere MQ Messages, GC34-6945, describes the user messages returned by WebSphere MQ, with explanations and suggested actions. It is designed for use as a quick reference.

This book is for system operators, system programmers, and anyone who needs to understand and take action in response to WebSphere MQ user messages.

WebSphere MQ Migration Information

WebSphere MQ Migration Information, SC34-6948, is for experienced users of WebSphere MQ who want to migrate their WebSphere MQ V5.3 systems to WebSphere MQ V6.0. It is specifically intended for system analysts, system programmers, system administrators, security administrators, network administrators, database administrators, and other users who have experience installing and managing WebSphere MQ. This book should be read with other publications in the WebSphere MQ V6.0 library describing specific tasks (installing, administering, and so on).

WebSphere MQ Programmable Command Formats and Administration Interface

WebSphere MQ Programmable Command Formats and Administration Interface, SC34-6942, starts by describing the facilities available in WebSphere MQ products for writing programs using the Programmable Command Formats (PCFs) to administer WebSphere MQ systems either locally or remotely. The second part of this book describes the administration interface for WebSphere MQ, known as the WebSphere MQ Administration Interface (MQAI). The MQAI is a programming interface that simplifies the use of PCF messages to configure WebSphere MQ.

This book is for system programmers who write programs to monitor and administer WebSphere MQ products. To understand the part about PCFs, you need experience in writing systems management applications, an understanding of the Message Queue Interface (MQI), and experience of WebSphere MQ programs in general, or familiarity with the content of the other books in the WebSphere MQ library. To understand the part about the MQAI, you need to understand the general concepts of WebSphere MQ and how to write programs in the C programming language or in Visual Basic for Windows.

WebSphere MQ Publish/Subscribe User's Guide

WebSphere MQ Publish/Subscribe User's Guide, SC34-6950, describes how to use WebSphere MQ Publish/Subscribe, which allows you to decouple the provider of information from the consumers of that information. WebSphere MQ Publish/Subscribe removes the need for your application to know anything about the target application. Similarly, the target application does not have to know anything about the source of the information it receives.

This book is for experienced users of WebSphere MQ who have a good knowledge of WebSphere MQ and want to use WebSphere MQ Publish/Subscribe. All the sample programs and header files are in the C programming language.

WebSphere MQ Queue Manager Clusters

WebSphere MQ Queue Manager Clusters, SC34-6933, describes how to organize, use and manage queue managers in virtual groups known as clusters. Clustering ensures that each queue manager within a given cluster knows about all the other queue managers in the same cluster. Clustering also makes the management of complex queue manager networks simpler.

This book is intended for:

- Network planners responsible for designing WebSphere MQ queue manager networks.
- Application programmers responsible for designing applications that access queues and queue managers within clusters.
- Systems administrators responsible for monitoring the local system and implementing planning details.
- System programmers responsible for designing and programming user exits.

You should understand the basic concepts of message queuing, for example the purpose of queues, queue managers, and channels. You should also be familiar with the WebSphere MQ products for the platforms you will be using, and the communications protocols used on those platforms. It will also be helpful if you understand how distributed queue management works.

WebSphere MQ Script (MQSC) Command Reference

WebSphere MQ Script (MQSC) Command Reference, SC34-6941, describes the MQSC commands, used by system operators and administrators to manage queue managers. It introduces the commands and tells you how to use them, before describing the commands in detail, in alphabetic order.

This book is intended for system programmers, system administrators, and system operators. To understand this book, you need to be familiar with the system facilities for the platform on which you are using WebSphere MQ product, and to understand the basic concepts of messaging and queuing.

WebSphere MQ Security

WebSphere MQ Security, SC34-6932, describes the factors you need to consider when planning to meet your security requirements in a WebSphere MQ environment. It provides the background information for you to evaluate the security provisions offered by WebSphere MQ and related products. This book also describes the Secure Sockets Layer (SSL) support in WebSphere MQ.

This book is for anyone responsible for planning or implementing security provisions to protect WebSphere MQ objects.

To understand this book, you do not need to have worked with message queuing products before, but you should understand the basic concepts of message queuing.

WebSphere MQ System Administration Guide

WebSphere MQ System Administration Guide, SC34-6928, describes the day-to-day management of local and remote WebSphere MQ objects. It includes topics such as security, recovery and restart, problem determination, and the dead-letter queue handler. It also includes the syntax of the WebSphere MQ control commands and tells you how to use installable services and exits to tailor your WebSphere MQ system.

This book is for system administrators and system programmers who manage the configuration and administration tasks for WebSphere MQ. It is also useful to application programmers who need to understand WebSphere MQ administration tasks. To understand this book, you need a good understanding of the operating systems it describes and of the utilities associated with them. You do not need to have worked with message queuing products before, but you should understand the basic concepts of message queuing.

WebSphere MQ Using C++

WebSphere MQ Using C++, SC34-6936, describes the C++ programming-language binding to the WebSphere MQ Message Queue Interface (MQI). It introduces the binding, describes considerations associated with using C++ with WebSphere MQ, and describes the WebSphere MQ C++ classes.

The book is for application programmers who write C++ programs that use the MQI. To understand this book, you need to know the C and C++ programming languages, understand the Booch methodology, understand the purpose of the Message Queue Interface (MQI), and have experience of WebSphere MQ programs in general, or familiarity with the content of other WebSphere MQ publications.

WebSphere MQ Using Java

WebSphere MQ Using Java, SC34-6935, is about WebSphere MQ classes for Java, which is also known as WebSphere MQ base Java, and WebSphere MQ classes for Java Message Service (JMS), which is also known as WebSphere MQ JMS. Both sets of classes are supplied with WebSphere MQ and are collectively known as WebSphere MQ Java.

WebSphere MQ JMS provides an implementation of Version 1.1 of Sun's JMS API specification. WebSphere MQ base Java, however, is a different set of classes and is not an implementation of the JMS API.

Using WebSphere MQ base Java or WebSphere MQ JMS, a Java application can connect to a WebSphere MQ queue manager and access its resources. A WebSphere MQ JMS publish/subscribe application can also connect directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker.

The book describes how to perform these tasks:
- Configuring WebSphere MQ Java after installation.

- Verifying the installation of WebSphere MQ Java.
- Administering WebSphere MQ JMS using the administration tool.
- Writing WebSphere MQ base Java and WebSphere MQ JMS applications.

The book also contains a reference sections that define the WebSphere MQ base Java and WebSphere MQ JMS classes and interfaces.

WebSphere MQ platform-specific publications

Each WebSphere MQ product is documented in at least one platform-specific publication, in addition to the WebSphere MQ family books. This section describes those publications, organized by platform.

For each WebSphere MQ cross-platform publication you will find the title and order number, followed by a brief description of the content of the publication and the intended audience, to help you decide whether you need that publication.

WebSphere MQ for AIX

WebSphere MQ for AIX Quick Beginnings

WebSphere MQ for AIX, V7.0 Quick Beginnings, GC34-6922, tells you how to plan for WebSphere MQ for AIX, how to install it, and how to verify that the product has installed correctly. It contains information about both the WebSphere MQ for AIX server and the WebSphere MQ for AIX client.

This book is for anyone responsible for installing WebSphere MQ for AIX. To understand this book you need a general understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for HP-UX

WebSphere MQ for HP-UX Quick Beginnings

WebSphere MQ for HP-UX, V7.0 Quick Beginnings, GC34-6923, tells you how to plan for WebSphere MQ for HP-UX, how to install it and how to verify that the product has installed correctly. It contains information about both the WebSphere MQ for HP-UX server and the WebSphere MQ for HP-UX client.

This book is for anyone responsible for installing WebSphere MQ for HP-UX. To understand this book you need an understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for i5/OS

WebSphere MQ for i5/OS Quick Beginnings

WebSphere MQ for i5/OS Quick Beginnings, GC34-6925, tells you how to plan for WebSphere MQ for i5/OS, and then how to install it and verify that the installation has worked.

This book is for anyone responsible for installing WebSphere MQ for i5/OS. To understand this book you need a general understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for i5/OS System Administration Guide V7.0

WebSphere MQ for i5/OS System Administration Guide V7.0, SC34-6930, describes the system administration aspects of WebSphere MQ for i5/OS, and the services provided to support commercial messaging. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

This book is for system administrators and system programmers who manage the configuration and administration tasks for WebSphere MQ. It is also useful to application programmers who must have some understanding of WebSphere MQ administration tasks. To use this book, you need a good understanding of the IBM operating system for the i5/OS system, and of the utilities associated with it. You do not need to have worked with message queuing products before, but you need to understand the basic concepts of message queuing.

WebSphere MQ for i5/OS Application Programming Reference (ILE RPG) V7.0

WebSphere MQ for i5/OS Application Programming Reference (ILE RPG) V7.0, SC34-6943, describes in full the WebSphere MQ for i5/OS programming interface in the RPG-ILE programming language. It shows you how to design and develop executable applications for WebSphere MQ, and it describes the sample programs available to help you write your RPG programs.

This book is intended for:
- Designers of applications that use message queuing techniques.
- Programmers who implement these application designs.

To understand this book, you should know how to write programs in the RPG programming language. You do not need to understand how to develop message queuing applications.

WebSphere MQ for Linux

WebSphere MQ for Linux Quick Beginnings

WebSphere MQ for Linux, V7.0 Quick Beginnings, GC34-6924, tells you how to plan for WebSphere MQ for Linux, how to install it and how to verify that the product has installed correctly. It contains information about both the WebSphere MQ for Linux server and the WebSphere MQ for Linux client.

This book is for anyone responsible for installing WebSphere MQ for Linux. To understand this book you need an understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for Solaris

WebSphere MQ for Solaris Quick Beginnings

WebSphere MQ for Solaris, V7.0 Quick Beginnings, GC34-6921, tells you how to plan for WebSphere MQ for Solaris, how to install it and how to verify that the product has installed correctly. It contains information about both the WebSphere MQ for Solaris server and the WebSphere MQ for Solaris client.

This book is for anyone responsible for installing WebSphere MQ for Solaris. To understand this book you need an understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for Windows

WebSphere MQ for Windows Quick Beginnings

WebSphere MQ for Windows, V7.0 Quick Beginnings, GC34-6920, shows you how to plan for WebSphere MQ for Windows, how to install it and how to verify that the product has installed correctly. It contains information about both the WebSphere MQ for Windows server and the WebSphere MQ for Windows client.

This book is for anyone responsible for installing WebSphere MQ for Windows. To understand this book you need an understanding of the basic concepts of WebSphere MQ.

WebSphere MQ for Windows Using .NET

WebSphere MQ Using .NET, GC34-6949, describes the WebSphere MQ classes for .NET, which can be used to access WebSphere MQ systems. It shows how to transfer this knowledge to become productive with the WebSphere MQ classes for .NET programming interfaces.

This book is for application programmers who want to send and receive messages in the .NET environment and are already familiar with the procedural WebSphere MQ application programming interface described in the WebSphere MQ Application Programming Guide.

To understand this book, you need some knowledge of the .NET programming environment, understand the purpose of the Message Queue Interface (MQI), and have experience of WebSphere MQ programs in general, or familiarity with the content of other WebSphere MQ publications.

WebSphere MQ for Windows Using the Component Object Model Interface

WebSphere MQ for Windows, Using the Component Object Model Interface, SC34-6938 describes the WebSphere MQ Automation Classes for ActiveX. It tells you how to design and program your applications using the WebSphere MQ ActiveX components, and how to resolve problems using trace and reason codes. It describes in detail each of the automation classes, the ActiveX interface to the WebSphere MQ Administration interface, and the support provided by WebSphere MQ for the Microsoft Active Directory Service Interfaces (ADSI).

This book is for designers and programmers who want to use the WebSphere MQ component interfaces to develop WebSphere MQ applications that run under Windows applications, using ActiveX components.

To understand this book, you need some experience of using ActiveX components and some experience or knowledge of WebSphere MQ.

WebSphere MQ for z/OS

WebSphere MQ for z/OS Concepts and Planning Guide

WebSphere MQ for z/OS Concepts and Planning Guide V7.0, GC34-6926, describes the concepts of WebSphere MQ for z/OS and tells you how to plan your WebSphere MQ for z/OS systems.

This book is for planners of z/OS systems that will use WebSphere MQ message queuing techniques and system Programmers who have to install, customize, and operate WebSphere MQ for z/OS. To understand this his book, you need to be familiar with the basic concepts of CICS, IMS, and WebSphere MQ. If you are going to use queue-sharing groups, you will also need to know DB2® and the zSeries Coupling Facility.

WebSphere MQ for z/OS Licensed Program Specifications

WebSphere MQ for z/OS Licensed Program Specifications, GC33-1350, provides a summary of the function available in WebSphere MQ for z/OS, together with detailed information about hardware and software requirements and the license under which the product is available.

WebSphere MQ for z/OS Problem Determination Guide

WebSphere MQ for z/OS Problem Determination Guide V7.0, GC34-6944, helps you to determine the causes of WebSphere MQ for z/OS problems, resolve those problems, deal with the IBM support center, and handle APARs.

This book is for those responsible for solving problems with WebSphere MQ for z/OS systems and application programs. To understand this book, you need to be familiar with system programming concepts, z/OS diagnostic procedures, and the structure and function of the WebSphere MQ for z/OS subsystems at your site. You should also be familiar with the other systems used with WebSphere MQ for z/OS at your site, for example, CICS and IMS.

WebSphere MQ for z/OS V7.0 Program Directory

WebSphere MQ for z/OS Program Directory, GI13-0529, includes the installation instructions for WebSphere MQ for z/OS. It also identifies hardware, software and storage requirements, and lists prerequisite APARs. It is for system programmers responsible for installing and maintaining WebSphere MQ for z/OS.

WebSphere MQ for z/OS System Administration Guide

WebSphere MQ for z/OS System Administration Guide V7.0, SC34-6929, tells you how to operate WebSphere MQ for z/OS using commands, panels, and utilities, and how to write applications to administer WebSphere MQ. The latter part of the book deals with termination, recovery, and restart.

This book is for system programmers and system administrators. To understand this book, you need to be familiar with the basic concepts of CICS, IMS, the z/OS job control language (JCL), and the z/OS Time Sharing Option (TSO). If you intend to use queue-sharing groups, you also need to know DB2 and the z/OS Coupling Facility. If you want to write programs to administer WebSphere MQ, you need to know how to write programs in one of the supported languages: COBOL, C, C++, Assembler, or PL/I. You do not need to have written message-queuing programs previously.

WebSphere MQ for z/OS System Setup Guide

WebSphere MQ for z/OS System Setup Guide V7.0, SC34-6927, tells you how to customize WebSphere MQ for z/OS after you have installed it. It also tells you how to monitor system use and performance, and how to set up security.

This book is for system programmers, system administrators, and security administrators. To understand this book, you need to be familiar with the basic concepts of CICS, IMS, the z/OS job control language (JCL), and the z/OS Time Sharing Option (TSO). If you intend to use queue-sharing groups, you also need to know DB2 and the z/OS Coupling Facility.

WebSphere MQ for z/OS Messages and Codes

WebSphere MQ for z/OS Messages and Codes V7.0, GC34-6946, lists all the user messages and abend reason codes returned by WebSphere MQ for z/OS, with explanations and suggested responses. It is designed for use as a quick reference, and is linked with the WebSphere MQ for z/OS Problem Determination Guide which you should also consult if a message indicates that there is a WebSphere MQ problem.

This book is for system operators, system programmers, and anybody else who needs to understand and respond to WebSphere MQ user messages. To understand this book, you need to understand the types of message WebSphere MQ produces, the different places to which it sends these messages, and the different audiences they are intended to reach.

Softcopy books

The WebSphere MQ books are supplied in two softcopy formats. These are supplied with the WebSphere MQ product on all platforms.

IBM Eclipse Help System
You can view all of the books in the library through the help system in WebSphere MQ Explorer. They are also available as a stand-alone IBM Eclipse Help System which you can install separately from the WebSphere MQ Explorer.

Portable Document Format (PDF)
You can view and print PDF files using the Adobe Acrobat Reader. If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at: http://www.adobe.com

Product family Web site

The WebSphere MQ books are also available from the product family Web site at:

`http://www.ibm.com/software/integration/mqfamily/`

By following links from this Web site you can:
- Obtain latest information about the product family.
- Access the books in HTML and PDF formats.
- Download a WebSphere MQ SupportPac.

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

Y

Z

Printed in USA

GC34-6981-04