WebSphere MQ

System Administration Guide

SC34-6068-01

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix I, "Notices" on page 515.

Second edition (October 2002)

This is the second edition of this book that applies to WebSphere MQ. It applies to the following products:
- WebSphere MQ for AIX, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for Linux for Intel, V5.3
- WebSphere MQ for Linux for zSeries, V5.3
- WebSphere MQ for Solaris, V5.3
- WebSphere MQ for Windows, V5.3

and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this book

This book applies to the following WebSphere® MQ Version 5.3 products:
- WebSphere MQ for AIX®, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
- WebSphere MQ for Solaris, V5.3
- WebSphere MQ for Windows®, V5.3

Other WebSphere MQ Version 5.3 products (on iSeries™ and zSeries™) have their own *System Administration* Guides. See *WebSphere MQ Bibliography and Glossary* for more information.

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 Queue Interface or MQI, so that programs developed on one platform can readily be transferred to another.

This book describes the system administration aspects of the WebSphere MQ Version 5.3 products, and the services they provide 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.

Installation of WebSphere MQ is described in the following:
- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*
- *WebSphere MQ for Windows, V5.3 Quick Beginnings*

Post-installation configuration of a distributed queuing network is described in *WebSphere MQ Intercommunication*.

Who this book is for

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 need a good understanding of the operating systems described here 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.

Terms used in this book

In this book, the term **Version 5.3 products** means:
- WebSphere MQ for AIX, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
- WebSphere MQ for Solaris, V5.3
- WebSphere MQ for Windows, V5.3

The term **WebSphere MQ for UNIX® systems** means:
- WebSphere MQ for AIX, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
- WebSphere MQ for Solaris, V5.3

We also use the term **UNIX systems** as a general terms for the UNIX platforms.

The term **WebSphere MQ for Windows systems** means WebSphere MQ running on the Windows platforms:
- Windows NT®
- Windows 2000
- Windows XP

We also use the term **Windows systems** or just **Windows** as general terms for these Windows platforms.

Using WebSphere MQ for Windows

Examples in this book relevant to WebSphere MQ for Windows can use New Technology file system (NTFS), high performance file system (HPFS), or file allocation table (FAT) file names.

The examples are valid for all file-naming systems, the name being transformed if necessary when the FAT system is in use. Name transformation is described in "Understanding WebSphere MQ file names" on page 18.

Using WebSphere MQ for UNIX systems

When you use WebSphere MQ on UNIX systems, **MQCONN** sets up its own signal handler for the signals SIGSEGV and SIGBUS. User handlers for these signals are restored after every MQI call.

The remaining signals are handled differently.
- SIGINT
- SIGQUIT
- SIGFPE
- SIGTERM
- SIGHUP

If any handler for this group of signals receives an interrupt within an MQI call, the application handler must exit the application.

On non-threaded applications, WebSphere MQ uses the UNIX interval timer ITIMER_REAL to generate SIGALRM signals. Any previous SIGALRM handler and timer interval is saved on entry to MQI and restored on exit. Any timer interval set is therefore frozen while within MQI.

The calls MQCONN and MQCONNX

References in this book to the call **MQCONN** (connect queue manager) can be replaced by references to the call **MQCONNX** (connect queue manager extended); **MQCONNX** requires an additional parameter. For more information about these calls, see the *WebSphere MQ Application Programming Reference*.

What's new for this release

This section describes the new function, of interest to system administrators, that has been added for this release of WebSphere MQ.

A change of name

With the release of Version 5 Release 3, we have rebranded MQSeries® to show its close relationship with the IBM® WebSphere brand of products that enable e-business. Throughout this book, and the rest of the library, MQSeries is now known by its new name, WebSphere MQ.

Using installable services

WebSphere MQ installable services:

- Provide you with the flexibility of choosing whether to use components provided by WebSphere MQ products, or replace or augment them with others.
- Allow vendors to participate, by providing components that might use new technologies, without making internal changes to WebSphere MQ products.
- Allow WebSphere MQ to exploit new technologies faster and cheaper, and so provide products earlier and at lower prices.

Previously, information on installable services was in *MQSeries Programmable System Management* We have now moved them into this book, in Part 7, "WebSphere MQ installable services and the API exit" on page 327.

MQSeries Programmable System Management has been completely removed. Other information that used to be in it has moved into the following books:

- Event monitoring is now in *WebSphere MQ Event Monitoring*.
- Programmable command formats are now in *WebSphere MQ Programmable Command Formats and Administration Interface*.

Using 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.

In Chapter 23, "API exits" on page 419, this book explains why you might want to use API exits, describes what administration tasks are involved in enabling them, and gives a brief introduction to writing API exits. For more detailed information about writing API exits, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

Using the Microsoft® Cluster Server (MSCS)

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

Chapter 13, “Supporting the Microsoft Cluster Server (MSCS)” on page 179 provides detailed information on how to set up WebSphere MQ to take advantage of MSCS support.

Using the Microsoft Transaction Server (MTS)

WebSphere MQ support for Microsoft Transaction Server (MTS) enables MTS business objects to use WebSphere MQ. MTS helps users run business logic applications in a typical middle tier server. MTS divides work up into activities, which are (typically) short independent chunks of business logic, such as *transfer funds from account A to account B*.

“Using the Microsoft Transaction Server (MTS)” on page 165 provides information on how to set up WebSphere MQ to take advantage of MTS support.

Secure Sockets Layer (SSL) support

The Secure Sockets Layer (SSL) protocol provides channel security, with protection against eavesdropping, tampering, and impersonation.

SSL is an industry-standard protocol that provides a data security layer between application protocols and the communications layer, usually TCP/IP. The SSL protocol was designed by the Netscape Development Corporation, and is widely deployed in both Internet applications and intranet applications. SSL defines methods for data encryption, server authentication, message integrity, and client authentication for a TCP/IP connection.

SSL uses public key and symmetric techniques to provide the following security services:

Message privacy
SSL uses a combination of public-key and symmetric-key encryption to ensure message privacy. Before exchanging messages, an SSL server and an SSL client perform an electronic handshake during which they agree to use a session key and an encryption algorithm. All messages sent between the client and the server are then encrypted. Encryption ensures that the message remains private even if eavesdroppers intercept it.

Message integrity
SSL uses the combination of a shared secret key and message hash functions. This ensures that nothing changes the content of a message as it travels between client and server.

Mutual authentication
During the initial SSL handshake, the server uses a public-key certificate to convince the client of the server’s identity. Optionally, the client can also exchange a public-key certificate with the server to ensure the authenticity of the client.

This book describes changes to the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands to support the SSL authentication information object. It also describes the **amqmcert** command for managing certificates on Windows systems, and the IKEYCMD interface for managing keys and certificates on UNIX systems.

“Protecting channels with SSL” on page 129 introduces the main features of SSL. Support for SSL introduces several new MQSC commands and queue manager

attributes. See the *WebSphere MQ Script (MQSC) Command Reference* for details of these. For a comprehensive review of security in WebSphere MQ including information on SSL, see *WebSphere MQ Security*.

Dumping authorities

A new command, **dmpmqaut** command lets you dump the current authorizations to a specified object. We describe it in "dmpmqaut (dump authority)" on page 265.

OAM generic profiles

OAM generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **setmqaut** commands against each individual object when it is created. "Using OAM generic profiles" on page 123 provides detailed information on using generic profiles.

Installable services

The installable services information, previously available in the *Programmable System Management* book, is now included in this book, starting at Part 7, "WebSphere MQ installable services and the API exit" on page 327.

Setting your license units

To fulfil the conditions of your license agreement with IBM, you must have purchased sufficient *license units* for the number of processors on which you are running WebSphere MQ. The installation dialog checks for this when you install WebSphere MQ

This book describes commands you can use to set (**setmqcap**) and display (**dspmqcap**) your license units (see "setmqcap (set license units)" on page 308 and "dspmqcap (display license units)" on page 275).

Part 1. Introduction

Chapter 1. Introduction to WebSphere MQ

This chapter introduces the WebSphere MQ Version 5.3 products from an administrator's perspective, and describes the basic concepts of WebSphere MQ and messaging. It contains these sections:

WebSphere MQ and message queuing

WebSphere MQ allows application programs to use *message queuing* to participate in message-driven processing. Application programs can communicate across different platforms by using the appropriate message queuing software products. For example, HP-UX and z/OS® applications can communicate through WebSphere MQ for HP-UX 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 known as the *message queue interface* (or MQI) wherever the applications run. This makes it easier for you to port application programs from one platform to another.

WebSphere MQ also provides a high level message handling API that makes it easier for programmers to deploy an intelligent network for business process integration within and between enterprises. This is called the *Application Messaging Interface* (AMI). The AMI moves many message-handling functions normally performed by messaging applications into the middleware layer, where a set of policies defined by the enterprise is applied on the application's behalf.

The MQI and AMI are described in detail in the *WebSphere MQ Application Programming Reference*.

Time-independent applications

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

Message-driven processing

When messages arrive on a queue, they can automatically start an application using *triggering*. If necessary, the applications can be stopped when the message (or messages) have been processed.

Messages and queues

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

What is a message?

A *message* is a string of bytes that is meaningful to the applications that use it. Messages are used to transfer information from one application program to another (or between 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.* The content and structure of the application data is defined by the application programs that use it.
- *A message descriptor.* The message descriptor identifies the message and contains additional 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 a complete description of the message descriptor, see the *WebSphere MQ Application Programming Reference* manual.

Message lengths

The default maximum message length is 4 MB, although you can increase this to a maximum length of 100 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length might 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 the queue
- The maximum message length defined by either the sending or receiving application
- The amount of storage available for the message

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

How do applications send and receive messages?

Application programs send and receive messages using **MQI calls**.

For example, to put a message onto a queue, an application:

1. Opens the required queue by issuing an MQI **MQOPEN** call
2. Issues an MQI **MQPUT** call to put the message onto the queue

Another application can retrieve the message from the same queue by issuing an MQI **MQGET** call

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

What is a queue?

A *queue* is a data structure used to store messages.

Each queue is owned by a *queue manager*. The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues. The messages might be put on the queue by application programs, or by a queue manager as part of its normal operation.

WebSphere MQ Version 5.3 supports queues over 2 GB in size; “Enabling large queues” on page 48 discusses this in more detail. For information about planning

the amount of storage you need for queues, see the *Quick Beginnings* guide for your platform, or visit the WebSphere MQ Web site for platform-specific performance reports:

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

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

- **Predefined queues** are created by an administrator using the appropriate MQSC or PCF commands. Predefined queues are permanent; they exist independently of the applications that use them and survive WebSphere MQ restarts.
- **Dynamic queues** are created when an application issues an **MQOPEN** request specifying the name of a *model queue*. The queue created is based on a *template queue definition*, which is called a model queue. You can create a model queue using the MQSC command DEFINE QMODEL. The attributes of a model queue (for example, the maximum number of messages that can be stored on it) are inherited by any dynamic queue that is created from it.

  Model queues have an attribute that specifies whether the dynamic queue is to be permanent or temporary. Permanent queues survive application and queue manager restarts; temporary queues are lost on restart.

Retrieving messages from queues

Suitably authorized applications can retrieve messages from a queue according to the following retrieval algorithms:

- First-in-first-out (FIFO).
- Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis.
- A program request for a specific message.

The **MQGET** request from the application determines the method used.

Objects

Many of the tasks described in this book involve manipulating WebSphere MQ **objects**. The object types are queue managers, queues, process definitions, channels, namelists, and authentication information objects.

The manipulation or *administration* of objects includes:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This is described in detail in *WebSphere MQ Intercommunication*.
- Creating *clusters* of queue managers to simplify the overall administration process, and to balance workload.

This book contains detailed information about administration in the following chapters:

- Chapter 2, “An introduction to WebSphere MQ administration” on page 15
- Chapter 3, “Managing queue managers using control commands” on page 23
- Chapter 4, “Administering local WebSphere MQ objects” on page 33
- Chapter 5, “Automating administration tasks” on page 55
- Chapter 6, “Administering remote WebSphere MQ objects” on page 59

- Chapter 7, "Administration using the WebSphere MQ Explorer" on page 75
- Chapter 8, "Administration using the WebSphere MQ Services snap-in" on page 81

Object names

The naming convention adopted for WebSphere MQ objects depends on the object.

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 is sent.

For the other types of object, each object has a name associated with it and can be referred to by that name. These names must be unique within one queue manager and object type. For example, you can have a queue and a process 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 *channels* which have a maximum of 20 characters. For more information about names, see "Names of WebSphere MQ objects" on page 243.

Managing objects

You can create, alter, display, and delete objects using:

- Control commands, which are typed in from a keyboard
- MQSC commands, which can be typed in from a keyboard or read from a file
- Programmable Command Format (PCF) messages, which can be used in an automation program
- WebSphere MQ Administration Interface (MQAI) calls in a program
- For WebSphere MQ for Windows only:
  - MQAI Component Object Model (COM) calls in a program
  - Active Directory Service interface (ADSI) calls in a program
  - The WebSphere MQ Explorer snap-in and WebSphere MQ Services snap-in running under the Microsoft Management Console (MMC)
  - The Windows NT Default Configuration application

For more information about these methods, see Chapter 2, "An introduction to WebSphere MQ administration" on page 15.

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; you can specify this attribute when you create a queue. The *DefinitionType* attribute specifies how the queue was created; you can only display this attribute.

In WebSphere MQ, there are two ways of referring to an attribute:

- Using its PCF name, for example, *MaxMsgLength*.
- Using its MQSC command name, for example, MAXMSGL.

This book mainly describes how to specify attributes using MQSC commands, and so it refers to most attributes using their MQSC command names, rather than their PCF names.

WebSphere MQ queues

There are four types of queue object available in WebSphere MQ.

Local queue object
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.

Remote queue object
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.

Before applications can send messages to a queue on another queue manager, you must have defined a transmission queue and channels between the queue managers, **unless** you have grouped one or more queue managers together into a *cluster*. For more information about clusters, see "Remote administration using clusters" on page 60.

Alias queue object
An alias queue 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.

Model queue object
A model queue defines a set of queue attributes that are used as a template for creating a dynamic queue. Dynamic queues are created by the queue manager when an application issues an **MQOPEN** request specifying a queue name that is the name of a model queue. The dynamic queue that is created in this way is a local queue whose attributes are taken from the model queue definition. The dynamic queue name can be specified by the application, or the queue manager can generate the name and return it to the application.

Dynamic queues defined in this way can be temporary queues, which do not survive product restarts, or permanent queues, which do.

Defining queues

Queues are defined to WebSphere MQ using:
- The MQSC command DEFINE
- The PCF Create Queue command

The commands specify 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

For further details about defining queue objects, see the *WebSphere MQ Script (MQSC) Command Reference* or *WebSphere MQ Programmable Command Formats and Administration Interface*.

Queues used by WebSphere MQ

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

Initiation queues

Initiation queues are queues that are used in triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event might be generated when the number of messages on a queue reaches a predefined depth. This event causes the queue manager to put a trigger message on a specified initiation queue. This trigger message is retrieved by a *trigger monitor*, a special application that monitors an initiation queue. The trigger monitor then starts the application program that was specified in the trigger message.

If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager. See "Managing objects for triggering" on page 51 and "runmqtrm (start trigger monitor)" on page 301. For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

Transmission queues

Transmission queues are queues that temporarily store 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. These queues are also used in remote administration; see "Remote administration from a local queue manager" on page 61. For information about the use of transmission queues in distributed queuing, see *WebSphere MQ Intercommunication*.

Each queue manager can have a default transmission queue. When a queue manager that is not part of a cluster puts a message onto a remote queue, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use the default transmission queue.

Cluster transmission queues

Each queue manager within a cluster has a cluster transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. A definition of this queue is created by default when you define a queue manager.

A queue manager that is part of the cluster can send messages on the cluster transmission queue to any other queue manager that is in the same cluster.

During name resolution, the cluster transmission queue takes precedence over the default transmission queue.

When a queue manager is part of a cluster, the default action is to use the SYSTEM.CLUSTER.TRANSMIT.QUEUE, except when the destination queue manager is not part of the cluster.

Dead-letter queues
A dead-letter (undelivered-message) queue is a queue that 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.

For distributed queuing, define a dead-letter queue on each queue manager involved.

Command queues
The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSC commands for processing. These commands are then retrieved by a WebSphere MQ component called the command server. The command server validates the commands, passes the valid ones on for processing by the queue manager, and returns any responses to the appropriate reply-to queue.

A command queue is created automatically for each queue manager when that queue manager is created.

Reply-to queues
When an application sends a request message, the application that receives the message can send back a reply message to the sending application. This message is put on a queue, called a reply-to queue, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.

Event queues
Instrumentation events can be used to monitor queue managers independently of MQI applications.

When an instrumentation event occurs, the queue manager puts an event message on an event queue. This message can then be read by a monitoring application, which might inform an administrator or initiate some remedial action if the event indicates a problem.

Note: Trigger events are quite different from instrumentation events in that trigger events are not caused by the same conditions, and do not generate event messages.

For more information about instrumentation events, see *WebSphere MQ Event Monitoring*.

WebSphere MQ queue managers

A *queue manager* provides queuing services to applications, and manages the queues that belong to it. It ensures that:

- Object attributes are changed according to the commands received.
- Special events such as trigger events or instrumentation 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.

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 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 can exist on a remote machine across the network, or might exist on the same machine as the local queue manager. WebSphere MQ supports multiple queue managers on the same machine.

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

Process definitions

A *process definition* object defines an application that starts in response to a trigger event on a WebSphere MQ queue manager. See the "Initiation queues" entry under "Queues used by WebSphere MQ" on page 8 for more information.

The process definition attributes include the application ID, the application type, and data specific to the application.

Channels

Channels are objects that provide a communication path from one queue manager to another. Channels are used in distributed queuing to move messages from one queue manager to another. They shield applications from the underlying communications protocols. The queue managers might 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.

For information on channels and how to use them, see *WebSphere MQ Intercommunication*.

Clusters

In a traditional WebSphere MQ network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager, it must define a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A *cluster* is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network, without the need for transmission queue, channel, and remote queue definitions.

For information about clusters, see Chapter 6, "Administering remote WebSphere MQ objects" on page 59, and *WebSphere MQ Queue Manager Clusters*.

Namelists

A namelist is a WebSphere MQ object that contains a list of other WebSphere MQ objects. Typically, namelists are used by applications such as trigger monitors, where they are used to identify a group of queues. The advantage of using a namelist is that it is maintained independently of applications; it can be updated

without stopping any of the applications that use it. Also, if one application fails, the namelist is not affected and other applications can continue using it.

Namelists are also used with queue manager clusters to maintain a list of clusters referred to by more than one WebSphere MQ object.

Authentication information objects

The queue manager authentication information object forms part of WebSphere MQ support for Secure Sockets Layer (SSL) security. It provides the definitions needed to check certificate revocation lists (CRLs) using LDAP servers. CRLs allow Certification Authorities to revoke certificates that can no longer be trusted.

This book describes using the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands with the authentication information object. For an overview of SSL and the use of the authentication information objects, see *WebSphere MQ Security*.

System default objects

The *system default objects* are a set of object definitions that are created automatically whenever a queue manager is created. You can copy and modify any of these object definitions for use in applications at your installation.

Default object names have the stem SYSTEM.DEFAULT; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE, and the default receiver channel is SYSTEM.DEFAULT.RECEIVER. You cannot rename these objects; default objects of these names are required.

When you define an object, any attributes that you do not specify explicitly are copied from the appropriate default object. For example, if you define a local queue, those attributes that you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

See Appendix A, “System and default objects” on page 475 for more information about system defaults.

Clients and servers

WebSphere MQ supports client-server configurations for its applications.

A WebSphere MQ *client* is a component that allows an application running on a system to issue MQI calls to a queue manager running on another system. The output from the call is sent back to the client, which passes it back to the application.

A WebSphere MQ *server* is a queue manager that provides queuing services to one or more clients. All the WebSphere MQ objects, for example queues, exist only on the queue manager machine (the WebSphere MQ server machine), and not on the client. A WebSphere MQ server can also support local WebSphere MQ applications.

The difference between a WebSphere MQ 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 *WebSphere MQ Intercommunication*.

For information about client support in general, see *WebSphere MQ Clients*.

WebSphere MQ applications in a client-server environment

When linked to a server, client WebSphere MQ applications can issue most 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 *WebSphere MQ Clients* for further information.

Extending queue manager facilities

The facilities provided by a queue manager can be extended by:
- User exits
- Installable services

User exits

User exits provide a mechanism for you to insert your own code into a queue manager function. The user exits supported include:

Channel exits
These exits change the way that channels operate. Channel exits are described in *WebSphere MQ Intercommunication*.

Data conversion exits
These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in the *WebSphere MQ Application Programming Guide*.

The cluster workload exit
The function performed by this exit is defined by the provider of the exit. Call definition information is given in *WebSphere MQ Queue Manager Clusters*.

Installable services

Installable services are more extensive than exits in that they have formalized interfaces (an API) with multiple entry points.

An implementation of an installable service is called a *service component*. You can use the components supplied with the WebSphere MQ product, or you can write your own component to perform the functions that you require.

Currently, the following installable services are provided:

Authorization service
The authorization service allows you to build your own security facility.

The default service component that implements the service is the Object Authority Manager (OAM). By default, the OAM is active, and you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM. For more information about the OAM, see Chapter 10, “WebSphere MQ security” on page 113.

Name service
The name service enables applications to share queues by identifying remote queues as though they were local queues.

A default service component that implements the name service is provided with WebSphere MQ. It uses the Open Software Foundation (OSF) Distributed Computing Environment (DCE). You can also write your own name service component. You might want to do this if you do not have DCE installed, for example. By default, the name service is inactive.

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. For more information about API exits, see Chapter 23, "API exits" on page 419 and the *WebSphere MQ Application Programming Guide*.

See Part 7, "WebSphere MQ installable services and the API exit" on page 327 for more information about the installable services.

Security

In WebSphere MQ, there are three methods of providing security:

- The Object Authority Manager (OAM) facility
- DCE security
- Channel security using Secure Sockets Layer (SSL)

Object Authority Manager (OAM) facility

Authorization for using MQI calls, commands, and access to objects is provided by the **Object Authority Manager** (OAM), which by default is enabled. Access to WebSphere MQ entities is controlled through WebSphere MQ user groups and the OAM. We provide a command line interface to enable administrators to grant or revoke authorizations as required.

For more information about creating authorization service components, see Chapter 10, "WebSphere MQ security" on page 113.

DCE security

Channel exits that use the DCE Generic Security Service (GSS) are provided by WebSphere MQ. For more information, see *WebSphere MQ Intercommunication*.

Channel security using SSL

The Secure Sockets Layer (SSL) protocol provides industry-standard channel security, with protection against eavesdropping, tampering, and impersonation.

SSL uses public key and symmetric techniques to provide message privacy and integrity and mutual authentication.

For a comprehensive review of security in WebSphere MQ including detailed information on SSL, see *WebSphere MQ Security*. For an overview of SSL, including pointers to the commands described in this book, see "Protecting channels with SSL" on page 129.

Transactional support

An application program can group a set of updates into a *unit of work*. These updates are usually logically related and must all be successful for data integrity to be preserved. If one update succeeds while another fails, data integrity is lost.

Transactional support

When a unit of work completes successfully, it is said to *commit*. Once committed, all updates made within that unit of work are made permanent and irreversible. However, if the unit of work fails, all updates are instead *backed out*. This process, where units of work are either committed or backed out with integrity, is known as *syncpoint coordination*.

A *local* unit of work is one in which the only resources updated are those of the WebSphere MQ queue manager. Here syncpoint coordination is provided by the queue manager itself using a single-phase commit process.

A *global* unit of work is one in which resources belonging to other resource managers, such as XA-compliant databases, are also updated. Here, a two-phase commit procedure must be used and the unit of work can be coordinated by the queue manager itself, or externally by another XA-compliant transaction manager such as IBM TXSeries®, or BEA Tuxedo.

For more information, see Chapter 11, "Transactional support" on page 139.

WebSphere MQ also provides support for the Microsoft Transaction Server (MTS). "Using the Microsoft Transaction Server (MTS)" on page 165 provides information on how to set up WebSphere MQ to take advantage of MTS support.

Chapter 2. An introduction to WebSphere MQ administration

This chapter introduces WebSphere MQ administration.

Administration tasks include creating, starting, altering, viewing, stopping, and deleting WebSphere MQ objects (queue managers, queues, clusters, processes, namelists, and channels).

The chapter contains the following sections:

Local and remote administration

You administer WebSphere MQ objects locally or remotely.

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

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

On platforms other than Windows systems, some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log onto the remote system and issue the commands from there or create a process that can issue the commands for you. On Windows systems you can use the WebSphere MQ Services snap-in.

Chapter 6, "Administering remote WebSphere MQ objects" on page 59 describes the subject of remote administration in greater detail.

Performing administration tasks using commands

There are three sets of commands that you can use to administer WebSphere MQ: control commands, MQSC commands, and PCF commands. These command sets are available on all platforms covered by this book.

Control commands

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

They are described in Chapter 3, "Managing queue managers using control commands" on page 23.

WebSphere MQ Script (MQSC) commands

Use MQSC commands to manage queue manager objects, including the queue manager itself, channels, queues, and process definitions.

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

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

- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not actually run
- *Direct mode*, where the MQSC commands are run on a local queue manager
- *Indirect mode*, where the MQSC commands are run on a remote queue manager

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

MQSC commands are available on other platforms, including OS/400®, and z/OS. MQSC commands are summarized in Appendix E, "Comparing command sets" on page 489.

The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

See "Performing local administration tasks using MQSC commands" on page 34 for more information about using MQSC commands in local administration.

PCF commands

WebSphere MQ programmable command format (PCF) commands allow administration tasks to be programmed into an administration program. In this way you can create queues and process definitions, and change queue managers, from a program.

PCF commands cover the same range of functions provided by the MQSC commands. See "PCF commands" on page 55 for more information.

You can use the WebSphere MQ Administration Interface (MQAI) to obtain easier programming access to PCF messages. This is described in greater detail in "Using the MQAI to simplify the use of PCFs" on page 56.

Administration on WebSphere MQ for Windows

On WebSphere MQ for Windows you can perform administration tasks using:

- PCF commands, MQSC commands, and control commands, as described in "Performing administration tasks using commands" on page 15
- The WebSphere MQ Explorer snap-in and the WebSphere MQ Services snap-in applications, running under the Microsoft Management Console (MMC)
- The Windows Default Configuration application

WebSphere MQ also supports use of the Microsoft Cluster Server (MSCS).

Using the WebSphere MQ Explorer

The WebSphere MQ Explorer is an application that runs under the Microsoft Management Console (MMC). It provides a graphical user interface for controlling resources in a network. Using the WebSphere MQ Explorer, you can:

- Define and control various resources including queue managers, queues, channels, process definitions, client connections, namelists, and clusters.
- Start or stop a local queue manager and its associated processes.
- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.
- Check to see which applications, users, or channels have a particular queue open, from the queue status.

You can invoke the WebSphere MQ Explorer from the First Steps application, or from the Windows Start prompt.

See Chapter 7, "Administration using the WebSphere MQ Explorer" on page 75 for more information.

Using the WebSphere MQ Services snap-in

The WebSphere MQ Services snap-in is an application that runs under the MMC. It allows you to perform more advanced tasks, typically associated with setting up and fine tuning the working environment for WebSphere MQ. For example, you can:

- Start or stop a queue manager, either locally or on another Windows system.
- Change the default queue manager.
- Start or stop individual WebSphere MQ processes such as a channel initiator or listener.
- Start or stop the command server.
- Start or stop the service trace.
- Set a queue manager to start automatically when you start up your workstation.

For more information, see Chapter 8, "Administration using the WebSphere MQ Services snap-in" on page 81.

Using the Windows Default Configuration application

You can use the Windows Default Configuration program from the WebSphere MQ First Steps application to create a *starter* (or default) set of WebSphere MQ objects. A summary of the default objects created is listed in Table 36 on page 477.

Using the Microsoft Cluster Server (MSCS)

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

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

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

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

Chapter 13, "Supporting the Microsoft Cluster Server (MSCS)" on page 179 provides detailed information on how to configure your WebSphere MQ for Windows system to use MSCS.

Understanding WebSphere MQ file names

Each WebSphere MQ queue, queue manager, namelist, and process object is represented by a file. Because object names are not necessarily valid file names, the queue manager converts the object name into a valid file name where necessary.

The default path to a queue manager directory is as follows:

- A prefix, which is defined in the WebSphere MQ configuration information:
  - In WebSphere MQ for Windows, the default prefix is `C:\Program Files\IBM\WebSphere MQ`. Use the WebSphere MQ Services snap-in if you want to change this. It is set on the All Queue Managers page of the WebSphere MQ properties.
  - In WebSphere MQ for UNIX systems, the default prefix is `/var/mqm`. You can change this in the `DefaultPrefix` stanza of the mqs.ini configuration file.
- The queue manager name is transformed into a valid directory name. For example, the queue manager:

  ```
  queue.manager
  ```

  would be represented as:

  ```
  queue!manager
  ```

  This process is referred to as *name transformation*.

Queue manager name transformation

In WebSphere MQ, you can give a queue manager a name containing up to 48 characters.

For example, you could name a queue manager:

```
QUEUE.MANAGER.ACCOUNTING.SERVICES
```

However, each queue manager is represented by a file and there are limitations on the maximum length of a file name, and on the characters that can be used in the name. As a result, the names of files representing objects are automatically transformed to meet the requirements of the file system.

The rules governing the transformation of a queue manager name are as follows:

1. Transform individual characters:
   · becomes !
   / becomes &
2. If the name is still not valid:
   a. Truncate it to eight characters
   b. Append a three-character numeric suffix

For example, assuming the default prefix and a queue manager with the name `queue.manager`:

- In WebSphere MQ for Windows with NTFS or FAT32, the queue manager name becomes:

  ```
  c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!manager
  ```

- In WebSphere MQ for Windows with FAT, the queue manager name becomes:

  ```
  c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!ma
  ```

- In WebSphere MQ for UNIX systems, the queue manager name becomes:

  ```
  /var/mqm/qmgrs/queue!manager
  ```

The transformation algorithm also distinguishes between names that differ only in case on file systems that are not case sensitive.

Object name transformation

Object names are not necessarily valid file system names. You might need to transform your object names. The method used is different from that for queue manager names because, although there are only a few queue manager names on each machine, there can be a large number of other objects for each queue manager. Process definitions, queues, and namelists are represented in the file system; channels are not affected by these considerations.

When a new name is generated by the transformation process, there is no simple relationship with the original object name. You can use the **dspmqfls** command to convert between real and transformed object names.

Part 2. Using commands to administer WebSphere MQ

Chapter 3. Managing queue managers using control commands

This chapter tells you how to perform operations on queue managers, command servers, and channels using control commands.

It contains the following topics:

Using control commands

You use control commands to perform operations on queue managers, command servers, and channels. Control commands can be divided into three categories, as shown in Table 1.

Table 1. Categories of control commands

Category	Description
Queue manager commands	Queue manager control commands include commands for creating, starting, stopping, and deleting queue managers and command servers
Channel commands	Channel commands include commands for starting and ending channels and channel initiators
Utility commands	Utility commands include commands associated with: • Running MQSC commands • Conversion exits • Authority management • Recording and recovering media images of queue manager resources • Displaying and resolving transactions • Trigger monitors • Displaying the file names of WebSphere MQ objects

For information about administration tasks for channels, see *WebSphere MQ Intercommunication*.

Using control commands on Windows systems

In WebSphere MQ for Windows, you enter control commands at a command prompt. In these environments, control commands and their flags are not case sensitive, but arguments to those commands (such as queue names and queue-manager names) are case sensitive.

For example, in the command:

```
crtmqm /u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
```

- The command name can be entered in uppercase or lowercase, or a mixture of the two. These are all valid: `crtmqm`, `CRTMQM`, and `CRTmqm`.

- The flag can be entered as `-u`, `-U`, `/u`, or `/U`.
- `SYSTEM.DEAD.LETTER.QUEUE` and `jupiter.queue.manager` must be entered exactly as shown.

For more information, see Chapter 16, "How to use WebSphere MQ control commands" on page 243.

Using the WebSphere MQ Explorer on Windows systems

On WebSphere MQ for Windows, you can use the WebSphere MQ Explorer to perform the operations described in this chapter, except for:

- Making an existing queue manager the default
- Preemptive shutdown

The tables in Appendix E, "Comparing command sets" on page 489 summarize which control commands have an equivalent WebSphere MQ Explorer implementation.

Using control commands on UNIX systems

In WebSphere MQ for UNIX systems, you enter control commands in a shell window. In these environments, control commands, including the command name itself, the flags, and any arguments, are case sensitive. For example, in the command:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
```

- The command name must be `crtmqm`, not `CRTMQM`.
- The flag must be `-u`, not `-U`.
- The dead-letter queue is called `SYSTEM.DEAD.LETTER.QUEUE`.
- The argument is specified as `jupiter.queue.manager`, which is different from `JUPITER.queue.manager`.

Take care to type the commands exactly as you see them in the examples.

For more information about the **crtmqm** command, see "crtmqm (create queue manager)" on page 259.

Creating a queue manager

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.

Before you can do anything with messages and queues, you must create and start at least one queue manager and its associated objects. To create a queue manager, use the WebSphere MQ control command **crtmqm** (described in "crtmqm (create queue manager)" on page 259). The **crtmqm** command automatically creates the required default objects and system objects (described in "System default objects" on page 11). Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. When you have created a queue manager and its objects, use the **strmqm** command to start the queue manager.

Note: WebSphere MQ does not support machine names that contain spaces. If you install WebSphere MQ on a computer with a machine name that contains spaces, you cannot create any queue managers.

Guidelines for creating queue managers

Before you can create a queue manager, there are several points you need to consider (especially in a production environment). Work through the following checklist:

Naming conventions

Use uppercase names so that you can communicate with queue managers on all platforms. Remember that names are assigned exactly as you enter them. To avoid the inconvenience of lots of typing, do not use unnecessarily long names.

Specify a unique queue manager name

When you create a queue manager, ensure that no other queue manager has the same name *anywhere* in your network. Queue manager names are not checked when the queue manager is created, and names that are not unique prevent you from creating channels for distributed queuing.

One way 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 can 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 communication with other enterprises, you can also include your own enterprise name as a prefix. This is not done in the examples, because it makes them more difficult to follow.

Note: Queue manager names in control commands are case-sensitive. This means that you are allowed to create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. However, it is better to avoid such complications.

Limit the number of queue managers

You can create as many queue managers as resources allow. However, because each queue manager requires its own resources, it is generally better to have one queue manager with 100 queues on a node than to have ten queue managers with ten queues each.

In production systems, many nodes are run with a single queue manager, but larger server machines might run with multiple queue managers.

Specify a default queue manager

Each node should have a default queue manager, though it is possible to configure WebSphere MQ on a node without one. The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an **MQCONN** call. It is also the queue manager that processes MQSC commands when you invoke the **runmqsc** command without specifying a queue manager name.

Specifying a queue manager as the default *replaces* any existing default queue manager specification for the node.

Changing the default queue manage can affect other users or applications. The change has no effect on currently-connected applications, because they can use the handle from their original connect call in any further MQI calls. This handle ensures that the calls are directed to the same queue manager. Any applications connecting *after* you have changed the default

queue manager connect to the new default queue manager. This might be what you intend, but you should take this into account before you change the default.

Creating a default queue manager is described in "Creating a default queue manager" on page 27.

Specify a dead-letter queue

The dead-letter queue is a local queue where messages are put if they cannot be routed to their intended destination.

It is important to have a dead-letter queue on each queue manager in your network. If you do not define one, errors in application programs might cause channels to be closed, and replies to administration commands might not be received.

For example, if an application tries to put a message on a queue on another queue manager, but gives the wrong queue name, 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.

When you create a queue manager, use the -u flag to specify the name of the dead-letter queue. You can also use an MQSC command to alter the attributes of a queue manager that you have already defined to specify the dead-letter queue to be used. See "Altering queue manager attributes" on page 42 for an example of the MQSC command ALTER.

Specify a default transmission queue

A transmission queue is a local queue on which messages in transit to a remote queue manager are queued before transmission. The default transmission queue is the queue that is used when no transmission queue is explicitly defined. Each queue manager can be assigned a default transmission queue.

When you create a queue manager, use the -d flag to specify the name of the default transmission queue. This does not actually create the queue; you have to do this explicitly later on. See "Working with local queues" on page 42 for more information.

Specify the logging parameters you require

You can specify logging parameters on the **crtmqm** command, including the type of logging, and the path and size of the log files.

In a development environment, the default logging parameters should be adequate. However, you can change the defaults if, for example:

- You have a low-end system configuration that cannot support large logs.
- You anticipate a large number of long messages being on your queues at the same time.
- You anticipate a lot of persistent messages passing through the queue manager.

Once you have set the logging parameters, some of them can only be changed by deleting the queue manager and re-creating it with the same name but with different logging parameters.

For more information about logging parameters, see Chapter 14, "Recovery and restart" on page 195.

Installing multiple queue managers

If you install multiple queue managers with different logical volumes for each queue manager, the `ftok` function might cause shared memory problems. The `ftok` function calculates the shared memory key from the node and the minor number of the housing logical volume. On AIX, shared memory problems can occur if two queue managers are installed in an HACMP environment with different logical volumes that have identical minor device numbers.

These shared memory problems do not occur if the different logical volumes are created such that they have different minor device numbers.

Creating a default queue manager

You create a default queue manager using the **crtmqm** command with the **-q** flag. The following **crtmqm** command:

- Creates a default queue manager called `SATURN.QUEUE.MANAGER`
- Creates the default and system objects
- Specifies the names of both a default transmission queue and a dead-letter queue

```
crtmqm -q -d MY.DEFAULT.XMIT.QUEUE -u SYSTEM.DEAD.LETTER.QUEUE  SATURN.QUEUE.MANAGER
```

where:

-q Indicates that this queue manager is the default queue manager.

-d MY.DEFAULT.XMIT.QUEUE
Is the name of the default transmission queue to be used by this queue manager.

Note: WebSphere MQ does not create a default transmission queue for you; you have to define it yourself.

-u SYSTEM.DEAD.LETTER.QUEUE
Is the name of the default dead-letter queue created by WebSphere MQ on installation.

SATURN.QUEUE.MANAGER
Is the name of this queue manager. This must be the last parameter specified on the **crtmqm** command.

The complete syntax of the **crtmqm** command is shown in "crtmqm (create queue manager)" on page 259.

The system and default objects are listed in Appendix A, "System and default objects" on page 475.

For WebSphere MQ for UNIX systems only

You can create the queue manager directory `/var/mqm/qmgrs/<qmgr>`, even on a separate local file system, before you use the **crtmqm** command. When you use **crtmqm**, if the `/var/mqm/qmgrs/<qmgr>` directory exists, is empty, and is owned by mqm, it is used for the queue manager data. If the directory is not owned by mqm, the creation fails with a First Failure Support Technology™ (FFST™) message. If the directory is not empty, a new directory is created.

Making an existing queue manager the default

You can make an existing queue manager the default queue manager. The way you do this depends on the platform you are using.

Windows systems

On Windows systems, use the WebSphere MQ Services snap-in to display the properties of the queue manager, and check the **Make queue manager the default** box. You need to stop and restart the queue manager for the change to take effect.

UNIX systems

When you create a default queue manager, its name is inserted in the `Name` attribute of the `DefaultQueueManager` stanza in the WebSphere MQ configuration file (mqs.ini). The stanza and its contents are automatically created if they do not exist.

- To make an existing queue manager the default, change the queue manager name on the `Name` attribute to the name of the new default queue manager. You must do this manually, using a text editor.
- If you do not have a default queue manager on the node, and you want to make an existing queue manager the default, create the *`DefaultQueueManager`* stanza with the required name yourself.
- If you accidentally make another queue manager the default and want to revert to the original default queue manager, edit the `DefaultQueueManager` stanza in mqs.ini, replacing the unwanted default queue manager with that of the one you want.

See Chapter 9, “Configuring WebSphere MQ” on page 89 for information about configuration files.

When you have changed the configuration information, stop the queue manager and restart it. See “Stopping a queue manager” on page 29 for information about how to do this.

Backing up configuration files after creating a queue manager

WebSphere MQ configuration information is stored in the Registry for Windows systems, and in configuration files on UNIX systems.

Windows systems

If you use WebSphere MQ for Windows, configuration information is stored in the Windows Registry. Use the WebSphere MQ Services snap-in (see “Changing configuration information on Windows systems” on page 89) or the **amqmdain** command (see “amqmdain (WebSphere MQ services control)” on page 253) to make changes to the Registry.

UNIX systems

There are two types of configuration file:

- When you install the product, the WebSphere MQ configuration file (mqs.ini) is created. It contains a list of queue managers that is updated each time you create or delete a queue manager. There is one mqs.ini file per node.
- When you create a new queue manager, a new queue manager configuration file (qm.ini) is automatically created. This contains configuration parameters for the queue manager.

After creating a queue manager, we recommend that you back up your configuration files.

If, later on, you create another queue manager that causes you problems, you can reinstate the backups when you have removed the source of the problem. As a general rule, back up your configuration files each time you create a new queue manager.

For more information about configuration files, see Chapter 9, "Configuring WebSphere MQ" on page 89.

Starting a queue manager

Although you have created a queue manager, it cannot process commands or MQI calls until you start it. You do this using the **strmqm** command as follows:

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

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

Starting a queue manager automatically

In WebSphere MQ for Windows, you can invoke a queue manager automatically when the system starts using the WebSphere MQ Services snap-in. See Chapter 8, "Administration using the WebSphere MQ Services snap-in" on page 81 for more information.

Stopping a queue manager

Use the **endmqm** command to stop a queue manager. For example, to stop a queue manager called `saturn.queue.manager`, type:

```
endmqm saturn.queue.manager
```

Quiesced shutdown

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

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

```
endmqm -c saturn.queue.manager
```

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

However, if you issue:

```
endmqm -w saturn.queue.manager
```

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

Immediate shutdown

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

For an immediate shutdown, type:

Stopping a queue manager

```
endmqm -i saturn.queue.manager
```

Preemptive shutdown

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

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

```
endmqm -p saturn.queue.manager
```

This stops all queue managers immediately.

If this method still does not work, see "Stopping a queue manager manually" on page 485 for an alternative solution.

For a detailed description of the **endmqm** command and its options, see "endmqm (end queue manager)" on page 283.

If you have problems shutting down a queue manager

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

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

If a problem occurs when you stop the queue manager, you can break out of the **endmqm** command using Ctrl-C.

You can then issue another **endmqm** command, but this time with a flag that specifies the type of shutdown that you require.

Restarting a queue manager

To restart a queue manager, type:

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

Restarting a queue manager is the same as starting it. In the WindowsExplorer and Services snap-ins, the only options are Start and Stop. Use the Start command to restart a queue manager.

Deleting a queue manager

To delete a queue manager, first stop it, then issue the following command:

```
dltmqm saturn.queue.manager
```

Notes:

1. Deleting a queue manager is a drastic step, because you also delete all resources associated with the queue manager, including all queues and their

messages and all object definitions. There is no displayed prompt that allows you to change your mind; when you press the Enter key all the associated resources are lost.

2. In WebSphere MQ for Windows, the **dltmqm** command also removes a queue manager from the automatic startup list (described in "Starting a queue manager automatically" on page 29). When the command has completed, a `WebSphere MQ queue manager ending` message is displayed; you are not told that the queue manager has been deleted.

For a description of the **dltmqm** command and its options, see "dltmqm (delete queue manager)" on page 263. Ensure that only trusted administrators have the authority to use this command. (For information about security, see Chapter 10, "WebSphere MQ security" on page 113.)

If this method for deleting a queue manager does not work, see "Removing queue managers manually" on page 486 for an alternative.

Chapter 4. Administering local WebSphere MQ objects

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

This chapter contains the following sections:

Supporting application programs that use the MQI

WebSphere MQ application programs need certain objects before they can run successfully. For example, Figure 1 shows an application that removes messages from a queue, processes them, and then sends some results to another queue on the same queue manager.

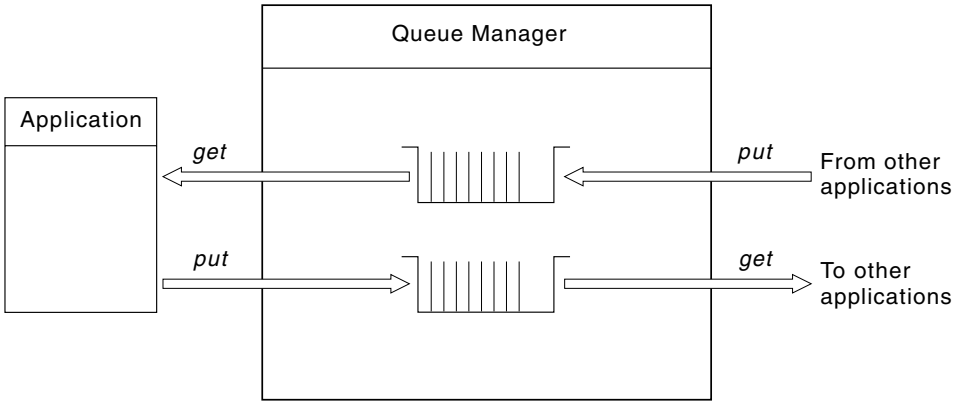


Figure 1. Queues, messages, and applications

Whereas applications can put messages onto local or remote queues (using **MQPUT**), they can only get messages directly from local queues (using **MQGET**).

Before this application can run, the following 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.

Application programs

- The application must be able to connect to the queue manager. To do this it must be linked to WebSphere MQ. See the *WebSphere MQ Application Programming Guide* for more information.
- 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 1 on page 33.

Performing local administration tasks using MQSC commands

This section introduces you to MQSC commands and tells you how to use them for some common tasks. If you use WebSphere MQ for Windows, you can also perform the operations described in this section using the WebSphere MQ Explorer. See Chapter 7, "Administration using the WebSphere MQ Explorer" on page 75 for more information.

You can use MQSC commands to manage queue manager objects, including the queue manager itself, clusters, channels, queues, namelists, process definitions, and authentication information objects. This section deals with queue managers, queues, and process definitions; for information about administering channel objects, see *WebSphere MQ Intercommunication*. For information about MQSC commands for managing the authentication information objects, see *WebSphere MQ Script (MQSC) Command Reference*.

You issue MQSC commands to a queue manager using the **runmqsc** command. (For details of this command, see "runmqsc (run MQSC commands)" on page 297.) You can do this interactively, issuing commands from a keyboard, or you can redirect the standard input device (`stdin`) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same. (For information about running the commands from a text file, see "Running MQSC commands from text files" on page 36.)

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

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

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

Object attributes specified in MQSC commands are shown in this book in uppercase (for example, RQMNAME), although they are not case sensitive. MQSC command attribute names are limited to eight characters. MQSC commands are available on other platforms, including OS/400 and z/OS.

MQSC commands are summarized in Appendix E, "Comparing command sets" on page 489.The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

WebSphere MQ object names

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

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

```
ORANGE.LOCAL.QUEUE
```

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

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

Case-sensitivity in MQSC commands

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

The **runmqsc** command invocation, in common with all WebSphere MQ control commands, is case sensitive in some WebSphere MQ environments. See "Using control commands" on page 23 for more information.

Standard input and output

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

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

Using MQSC commands interactively

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

```
runmqsc
```

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

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

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

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

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

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

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

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

Feedback from MQSC commands

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

```
AMQ8006: WebSphere MQ queue created.
```

This message confirms that a queue has been created.

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

AMQ8426: Valid MQSC commands are:


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

This message indicates that you have made a syntax error.

These messages are sent to the standard output device. If you have not entered the command correctly, refer to the *WebSphere MQ Script (MQSC) Command Reference* for the correct syntax.

Ending interactive input of MQSC commands

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

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

Running MQSC commands from text files

Running MQSC commands interactively is suitable for quick tests, but if you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a text file. (See “Standard input and output” on page 35

page 35 for information about stdin and stdout.) To do this, first create a text file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. For example, the following command runs a sequence of commands contained in the text file myprog.in:

```
runmqsc < myprog.in
```

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

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

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

This command invokes the MQSC commands contained in the MQSC command file myprog.in. Because we have not specified a queue manager name, the MQSC commands run against the default queue manager. The output is sent to the text file myprog.out. Figure 2 shows an extract from the MQSC command file myprog.in and Figure 3 on page 38 shows the corresponding extract of the output in myprog.out.

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

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

MQSC command files

MQSC commands are written in human-readable form, that is, in ASCII text. Figure 2 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

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

Figure 2. Extract from an MQSC command file

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

MQSC command reports

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

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

  ```
  Starting WebSphere MQ Commands.
  ```

- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in Figure 3. However, you can use the -e flag on the **runmqsc** command to suppress the output.
- A syntax error message for any commands found to be in error.
- An *operator message* indicating the outcome of running each command. For example, the operator message for the successful completion of a DEFINE QLOCAL command is:

  ```
  AMQ8006: WebSphere MQ queue created.
  ```

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

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

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

Figure 3. Extract from an MQSC command report file

Running the supplied MQSC command files

These MQSC command files are supplied with WebSphere MQ:

amqscos0.tst
: Definitions of objects used by sample programs.

amqscic0.tst
: Definitions of queues for CICS transactions.

In WebSphere MQ for Windows, these files are located in the directory `c:\Program Files\IBM\WebSphere MQ\tools\mqsc\samples`.

On UNIX systems these files are located in the directory `opt/mqm/samp` (`usr/mqm/samp` on AIX).

The command that runs them is:

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

Using runmqsc to verify commands

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

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

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

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

The returned report is similar to that shown in Figure 3 on page 38.

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

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

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

Running MQSC commands from batch files

If you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a batch file. To do this, first create a batch file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. The following example:

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

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

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

runmqsc $MYTEMPQM << EOF
  DEFINE CHANNEL(NTLM) CHLTYPE(SVRCONN) TRPTYPE(TCP) SCYEXIT('amqrspin(SCY_NTLM)')
  DEFINE CHANNEL(NTLM) CHLTYPE(CLNTCONN) QMNAME('$MYTEMPQM') CONNAME('127.0.0.1($MYPORT)')
  ALTER  CHANNEL(NTLM) CHLTYPE(CLNTCONN) SCYEXIT('amqrspin(SCY_NTLM)')
  DEFINE QLOCAL(TESTQ)
EOF

amqsputc TESTQ $MYTEMPQM << EOF
hello world
EOF

endmqm -i $MYTEMPQM
```

Figure 4. Example script for running MQSC commands from a batch file

Resolving problems with MQSC commands

If you cannot get MQSC commands to run, use the following information to see if any of these common problems apply to you. It is not always obvious what the problem is when you read the error generated.

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

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

  ```
  AMQ8118: WebSphere MQ queue manager does not exist.
  ```

- If you redirect output to a file, use the **>** redirection operator. By default, the file is put in the current working directory at the time **runmqsc** is invoked. Specify a fully-qualified file name to send your output to a specific file and directory.
- Check that you have created the queue manager that is going to run the commands.

  To do this on Windows systems, use the WebSphere MQ Explorer to display a list of queue managers. On UNIX systems, look in the WebSphere MQ configuration file, mqs.ini. This file contains the names of the queue managers and the name of the default queue manager, if you have one.
- The queue manager must be running. If it is not, start it; (see “Starting a queue manager” on page 29). You get an error message if you try to start a queue manager that is already running.
- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, or you get this error:

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

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

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

- You cannot enter MQSC commands before you issue the **runmqsc** command.

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

```
runmqsc
 .
 .
Starting WebSphere MQ Commands.

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

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

Working with queue managers

This section contains examples of some MQSC commands that you can use to display or alter queue manager attributes. See the *WebSphere MQ Script (MQSC) Command Reference* for detailed information about these commands.

Displaying queue manager attributes

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

```
DISPLAY QMGR
```

Typical output from this command is shown in Figure 5 on page 42.

```
0784726, 5639-B43 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Starting WebSphere MQ script Commands.


DISPLAY QMGR
     1 : DISPLAY QMGR
AMQ8408: Display Queue Manager details.
   DESCR( )                                DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
   DEFXMITQ( )                             CHADEXIT( )
   CLWLEXIT( )                             CLWLDATA( )
   REPOS( )                                REPOSNL( )
   SSLKEYR(C:\MQM\qmgrs\Q\ssl\key)         SSLCRLNL( )
   SSLCRYP( )                              COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
   QMNAME(saturn.queue.manager)            CRDATE(2002-03-14)
   CRTIME(17.21.40)                        ALTDATE(2002-03-14)
   ALTTIME(17.21.40)                       QMID(Q_2002-03-14_17.21.40)
   TRIGINT(999999999)                      MAXHANDS(256)
   MAXUMSGS(10000)                         AUTHOREV(DISABLED)
   INHIBTEV(DISABLED)                      LOCALEV(DISABLED)
   REMOTEEV(DISABLED)                      PERFMEV(DISABLED)
   STRSTPEV(ENABLED)                       CHAD(DISABLED)
   CHADEV(DISABLED)                        CLWLLEN(100)
   MAXMSGL(4194304)                        CCSID(437)
   MAXPRTY(9)                              CMDLEVEL(530)
   PLATFORM(WINDOWSNT)                     SYNCPT
   DISTL(YES)
```

Figure 5. Typical output from a DISPLAY QMGR command

The ALL parameter (the default) on the DISPLAY QMGR command displays all the queue manager attributes. In particular, the output tells you the default queue manager name (`saturn.queue.manager`), the dead-letter queue name (SYSTEM.DEAD.LETTER.QUEUE), and the command queue name (SYSTEM.ADMIN.COMMAND.QUEUE).

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

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

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

Altering queue manager attributes

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

```
runmqsc jupiter.queue.manager

ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
```

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

Working with local queues

This section contains examples of some MQSC commands that you can use to manage local, model, and alias queues. See the *WebSphere MQ Script (MQSC) Command Reference* for detailed information about these commands.

Defining a local queue

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

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

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

Using the MQSC command shown below, we define a queue called ORANGE.LOCAL.QUEUE, with the following characteristics:

- It is enabled for gets, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an *ordinary* queue; it is not an initiation queue or transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +
       DESCR('Queue for messages from other systems') +
       PUT (DISABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (FIFO) +
       MAXDEPTH (1000) +
       MAXMSGL (2000) +
       USAGE (NORMAL);
```

Notes:

1. Most of these attributes are the defaults as supplied with the product. We have shown them here for purposes of illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying default object attributes" on page 44.
2. USAGE (NORMAL) indicates that this queue is not a transmission queue.
3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 45.

Defining a dead-letter queue

We recommend that each queue manager has a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You must tell the queue manager about the dead-letter queue. You do this by specifying a dead-letter queue name on the **crtmqm** command (`crtmqm -u DEAD.LETTER.QUEUE`, for example), or by using the DEADQ attribute on the ALTER QMGR command to specify one later. You must define the dead-letter queue before using it.

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

A dead-letter queue has no special requirements except that:

- It must be a local queue

- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH)

WebSphere MQ provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see Chapter 12, "The WebSphere MQ dead-letter queue handler" on page 167.

Displaying default object attributes

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

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

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

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

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

This command displays the three specified attributes as follows:

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

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

Copying a local queue definition

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

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

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

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

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

This command copies the attributes of the queue ORANGE.LOCAL.QUEUE to the queue THIRD.QUEUE, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Notes:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you a define a local queue, without specifying LIKE, it is the same as DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).

Changing local queue attributes

You can change queue attributes in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE attribute. In "Defining a local queue" on page 43, we defined the queue called ORANGE.LOCAL.QUEUE. Suppose, for example, you want to increase the maximum message length on this queue to 10 000 bytes.

- Using the ALTER command:

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

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

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

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

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

Clearing a local queue

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

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

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

You cannot clear a queue if:

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

Deleting a local queue

Use the MQSC command DELETE QLOCAL to delete a local queue. A queue cannot be deleted if it has uncommitted messages on it. However, if the queue has one or more committed messages and no uncommitted messages, it can be deleted only if you specify the PURGE option. For example:

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

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

Browsing queues

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

In WebSphere MQ for Windows, the default file names and paths are:

Source

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\
```

Executable

```
c:\Program files\IBM\WebSphere MQ\tools\c\samples\bin\amqsbcg.exe
```

In MQSeries for UNIX systems, the default file names and paths are:

Source

`/opt/mqm/samp/amqsbcg0.c` (`/usr/mqm/samp/amqsbcg0.c` on AIX)

Executable

`/opt/mqm/samp/bin/amqsbcg` (`/usr/mqm/samp/bin/amqsbcg` on AIX)

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

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

Typical results from this command are shown in Figure 6 on page 47.

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

 MQOPEN - 'SYSTEM.ADMIN.QMGR.EVENT'


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

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

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

****   Message      ****

 length - 104 bytes

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

 No more messages
 MQCLOSE
 MQDISC
```

Figure 6. Typical results from queue browser

Monitoring local queues with the Windows Performance Monitor

Administrators of WebSphere MQ for Windows can monitor the performance of local queues using the Windows Performance Monitor.

The Performance Monitor displays a new object type called `WebSphere MQ Queues` in which performance data for local queues is stored.

Performance Monitor

Active local queues defined in running queue managers are displayed as `QueueName:QMName` in the Performance Monitor Instance list when you select the `WebSphere MQ Queues` object type. `QMName` denotes the name of the queue manager owning the queue, and `QueueName` denotes the name of the local queue.

For each queue, you can view information relating to the following:
- The current queue depth
- The queue depth as a percentage of the maximum queue depth
- The number of messages being placed on the queue each second
- The number of messages being removed from the queue each second

For messages sent to a distribution list, the Performance Monitor counts the number of messages being put onto each queue.

In the case of segmented messages, the Performance Monitor counts the appropriate number of small messages.

Performance data is obtained from statistical data maintained by the WebSphere MQqueue managers for each local queue. However, performance data is available only for queues that are accessed ***after*** the Performance Monitor has started.

You can monitor the performance of queues on computers other than that on which the Performance Monitor is running, by selecting your target computer from the Performance Monitor, which works using the Windows Network Neighborhood hierarchy.

Enabling large queues

WebSphere MQ Version 5.3 supports queues larger than 2 GB. On Windows systems, support for large files is available without any additional enablement. On AIX, HP-UX, Linux, and Solaris systems, you need to explicitly enable large file support before you can create queue files larger than 2 GB. See your operating system documentation for information on how to do this.

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

Working with alias queues

An alias queue provides a way of referring indirectly to another queue. The other queue can be either:
- A local queue (see "Defining an initiation queue" on page 52).
- A local definition of a remote queue (see "Creating a local definition of a remote queue" on page 66).

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

Alias queues are useful for:
- Giving different applications different levels of access authorities to the target queue.

- Allowing different applications to work with the same queue in different ways. (Perhaps you want to assign different default priorities or different default persistence values.)
- Simplifying maintenance, migration, and workload balancing. (Perhaps you want to change the target queue name without having to change your application, which continues to use the alias.)

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

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

Defining an alias queue

The following command creates an alias queue:

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

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

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

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

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

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

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

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

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

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

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

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

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

Using other commands with alias queues

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

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

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

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

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

Use the following command to delete this queue alias:

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

You cannot delete an alias queue if an application currently has the queue open. See the *WebSphere MQ Script (MQSC) Command Reference* for more information about this and other alias queue commands.

Working with model queues

A queue manager creates a *dynamic queue* if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A *model queue* is a template that specifies the attributes of any dynamic queues created from it.

Model queues provide a convenient method for applications to create queues as required.

Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes, except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not.) For example:

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
       DESCR('Queue for messages from application X') +
       PUT (DISABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (FIFO) +
       MAXDEPTH (1000) +
       MAXMSGL (2000) +
       USAGE (NORMAL) +
       DEFTYPE (PERMDYN)
```

This command creates a model queue definition. From the DEFTYPE attribute, you can see that the actual queues created from this template are permanent dynamic

queues. Any attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

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

Using other commands with model queues

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

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

```
DISPLAY QUEUE (GREEN.MODEL.QUEUE)
```

Use the following command to alter the model to enable puts on any dynamic queue created from this model:

```
ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)
```

Use the following command to delete this model queue:

```
DELETE QMODEL (RED.MODEL.QUEUE)
```

Managing objects for triggering

WebSphere MQ enables you to start an application automatically when certain conditions on a queue are met. For example, you might want to start an application when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *WebSphere MQ Application Programming Guide*.

This section tells you how to set up the required objects to support triggering on WebSphere MQ.

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC commands).

In this example, a trigger event is to be generated when there are 100 messages of priority 5 or greater on the local queue MOTOR.INSURANCE.QUEUE, as follows:

```
DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
       PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
       MAXMSGL (2000) +
       DEFPSIST (YES) +
       INITQ (MOTOR.INS.INIT.QUEUE) +
       TRIGGER +
       TRIGTYPE (DEPTH) +
       TRIGDPTH (100)+
       TRIGMPRI (5)
```

where:

QLOCAL (MOTOR.INSURANCE.QUEUE)
Is the name of the application queue being defined.

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
Is the name of the application to be started by a trigger monitor program.

MAXMSGL (2000)
Is the maximum length of messages on the queue.

DEFPSIST (YES)
Specifies that messages on this queue are persistent by default.

INITQ (MOTOR.INS.INIT.QUEUE)
Is the name of the initiation queue on which the queue manager is to put the trigger message.

TRIGGER
Is the trigger attribute value.

TRIGTYPE (DEPTH)
Specifies that a trigger event is generated when the number of messages of the required priority (TRIGMPRI) reaches the number specified in TRIGDPTH.

TRIGDPTH (100)
Is the number of messages required to generate a trigger event.

TRIGMPRI (5)
Is the priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue MOTOR.INS.INIT.QUEUE for guidance:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
       GET (ENABLED) +
       NOSHARE +
       NOTRIGGER +
       MAXMSGL (2000) +
       MAXDEPTH (1000)
```

Defining a process

Use the DEFINE PROCESS command to create a process definition. A process definition defines the application to be used to process messages from the application queue. The application queue definition names the process to be used and thereby associates the application queue with the application to be used to process its messages. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

```
DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
               DESCR ('Insurance request message processing') +
               APPLTYPE (UNIX) +
               APPLICID ('/u/admin/test/IRMP01') +
               USERDATA ('open, close, 235')
```

Where:

MOTOR.INSURANCE.QUOTE.PROCESS
Is the name of the process definition.

DESCR ('Insurance request message processing')
Describes the application program to which this definition relates. This text

is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPLTYPE (UNIX)
Is the type of application to be started.

APPLICID ('/u/admin/test/IRMP01')
Is the name of the application executable file, specified as a fully qualified file name. In Windows systems, a typical APPLICID value would be c:\appl\test\irmp01.exe.

USERDATA ('open, close, 235')
Is user-defined data, which can be used by the application.

Displaying attributes of a process definition

Use the DISPLAY PROCESS command to examine the results of your definition. For example:

```
DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
```

```
    24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
AMQ8407: Display Process details.
    DESCR ('Insurance request message processing')
    APPLICID ('/u/admin/test/IRMP01')
    USERDATA (open, close, 235)
    PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
    APPLTYPE (UNIX)
```

You can also use the MQSC command ALTER PROCESS to alter an existing process definition, and the DELETE PROCESS command to delete a process definition.

Chapter 5. Automating administration tasks

This chapter assumes that you have experience of administering WebSphere MQ objects.

You might decide that it would be beneficial to your installation to automate some administration and monitoring tasks. You can automate administration tasks for both local and remote queue managers using programmable command format (PCF) commands.

This chapter describes:

- How to use programmable command formats to automate administration tasks in PCF commands
- How to use the command server in "Managing the command server for remote administration" on page 64
- Support for Microsoft's Active Directory Service Interfaces (ADSI) in "Active Directory Services Interfaces" on page 57

PCF commands

The purpose of WebSphere MQ programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way you can create queues, process definitions, channels, and namelists, and change queue managers, from a program.

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

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

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

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

Message descriptor
This is a standard WebSphere MQ message descriptor, in which:

- Message type (*MsqType*) is MQMT_REQUEST.
- Message format (*Format*) is MQFMT_ADMIN.

Application data
Contains the PCF message including the PCF header, in which:

- The PCF message type (*Type*) specifies MQCFT_COMMAND.

- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

For a complete description of the PCF data structures and how to implement them, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

PCF object attributes

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

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

Using the MQAI to simplify the use of PCFs

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

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

Use the MQAI:

To simplify the use of PCF messages
The MQAI is an easy way to administer WebSphere MQ; you do not have to write your own PCF messages, avoiding the problems associated with complex data structures.

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

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

To implement self-administering applications and administration tools
For example, the Active Directory Services Interfaces provided by WebSphere MQ for Windows use the MQAI.

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

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

For more information about using the MQAI, and PCFs in general, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

Active Directory Services Interfaces

Active Directory Service Interfaces (ADSI) support allows client applications to use a common set of Component Object Model (COM) interfaces to communicate with, and control, any application that implements them.

Unlike tools written using other WebSphere MQ administration interfaces, those that use the ADSI are not limited to manipulating WebSphere MQ servers. The same tool can control Windows, Lotus® Notes™, or any application implementing the ADSI.

WebSphere MQ support for the ADSI is implemented through the use of the **IBMMQSeries namespace**.

Any programming language that supports the COM interfaces can be used to implement ADSI clients.

For more information about the ADSI, visit the Microsoft web site at:

```
www.microsoft.com
```

For more information about Component Object Model (COM) interfaces, see *WebSphere MQ for Windows, V5.3 Using the Component Object Model Interface*.

Client connection channels in the Active Directory

On Windows systems that support the Active Directory, WebSphere MQ publishes client connection channels in the Active Directory to provide dynamic client-server binding.

When CLNTCONN channels are defined, they are written into a binary file called AMQCLCHL.TAB. If the client channels use the TCP/IP protocol, the WebSphere MQ server also publishes them in the Active Directory. When the WebSphere MQ client determines how to connect to the server, it looks for a relevant CLNTCONN definition using the following search order:

1. MQSERVER environment variable
2. AMQCLCHL.TAB file
3. Active Directory

This order means that any current applications are not affected by the change. You can think of these entries in the Active Directory as records in the AMQCLCHL.TAB file, and the WebSphere MQ client processes them in the same way. To configure and administer support for publishing client connection channel definitions in the Active Directory, use the **setmqscp** command, as described in "setmqscp (set service connection points)" on page 311.

Chapter 6. Administering remote WebSphere MQ objects

This chapter tells you how to administer WebSphere MQ objects on a remote queue manager using MQSC commands, and how to use remote queue objects to control the destination of messages and reply messages.

This chapter describes:

Channels, clusters, and remote queuing

A queue manager communicates with another queue manager by sending a message and, if required, receiving back a response. The receiving queue manager could be:

- On the same machine
- On another machine in the same location (or even on the other side of the world)
- Running on the same platform as the local queue manager
- Running on another platform supported by WebSphere MQ

These messages might originate from:

- User-written application programs that transfer data from one node to another
- User-written administration applications that use PCF commands, the MQAI, or the ADSI
- Queue managers sending:
  - Instrumentation event messages to another queue manager
  - MQSC commands issued from a **runmqsc** command in indirect mode (where the commands are run on another queue manager)

Before a message can be sent to a remote queue manager, the local queue manager needs a mechanism to detect the arrival of messages and transport them consisting of:

- At least one channel
- A transmission queue
- A channel listener
- A channel initiator

A channel is a one-way communication link between two queue managers and can carry messages destined for any number of queues at the remote queue manager.

Each end of the channel has a separate definition. For example, if one end is a sender or a server, the other end must be a receiver or a requester. A simple channel consists of a *sender channel definition* at the local queue manager end and a *receiver channel definition* at the remote queue manager end. The two definitions must have the same name and together constitute a single channel.

Channels, clusters, and remote queuing

If you want the remote queue manager to respond to messages sent by the local queue manager, set up a second channel to send responses back to the local queue manager.

Use the MQSC command DEFINE CHANNEL to define channels. In this chapter, the examples relating to channels use the default channel attributes unless otherwise specified.

There is a message channel agent (MCA) at each end of a channel, controlling the sending and receiving of messages. The MCA takes messages from the transmission queue and puts them on the communication link between the queue managers.

A transmission queue is a specialized local queue that temporarily holds messages before the MCA picks them up and sends them to the remote queue manager. You specify the name of the transmission queue on a *remote queue definition*.

You can allow an MCA to transfer messages using multiple threads. This process is known as *pipelining*. Pipelining enables the MCA to transfer messages more efficiently, improving channel performance. See "Channels" on page 105 for details of how to configure a channel to use pipelining.

"Preparing channels and transmission queues for remote administration" on page 62 tells you how to use these definitions to set up remote administration.

For more information about setting up distributed queuing in general, see *WebSphere MQ Intercommunication*.

Remote administration using clusters

In a WebSphere MQ network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager, it must define a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A *cluster* is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network without complex transmission queue, channel, and queue definitions. Clusters can be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications. Even the smallest cluster reduces system administration overheads.

Establishing a network of queue managers in a cluster involves fewer definitions than establishing a traditional distributed queuing environment. With fewer definitions to make, you can set up or change your network more quickly and easily, and reduce the risk of making an error in your definitions.

To set up a cluster, you need one cluster sender (CLUSSDR) and one cluster receiver (CLUSRCVR) definition for each queue manager. You do not need any transmission queue definitions or remote queue definitions. The principles of remote administration are the same when used within a cluster, but the definitions themselves are greatly simplified.

For more information about clusters, their attributes, and how to set them up, refer to *WebSphere MQ Queue Manager Clusters*.

Remote administration from a local queue manager

This section tells you how to administer a remote queue manager from a local queue manager using MQSC and PCF commands.

Preparing the queues and channels is essentially the same for both MQSC and PCF commands. In this book, the examples show MQSC commands, because they are easier to understand. For more information about writing administration programs using PCF commands, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

You send MQSC commands to a remote queue manager either interactively or from a text file containing the commands. The remote queue manager might be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in other WebSphere MQ environments, including UNIX systems, Windows systems, OS/400, and z/OS.

To implement remote administration, you must create specific objects. Unless you have specialized requirements, you should find that the default values (for example, for maximum message length) are sufficient.

Preparing queue managers for remote administration

Figure 7 shows the configuration of queue managers and channels that you need for remote administration using the **runmqsc** command. The object `source.queue.manager` is the source queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned. The object `target.queue.manager` is the name of the target queue manager, which processes the commands and generates any operator messages.

Note: If you are using **runmqsc** with the -w option, `source.queue.manager` ***must*** be the default queue manager. For further information on creating a queue manager, see "crtmqm (create queue manager)" on page 259.

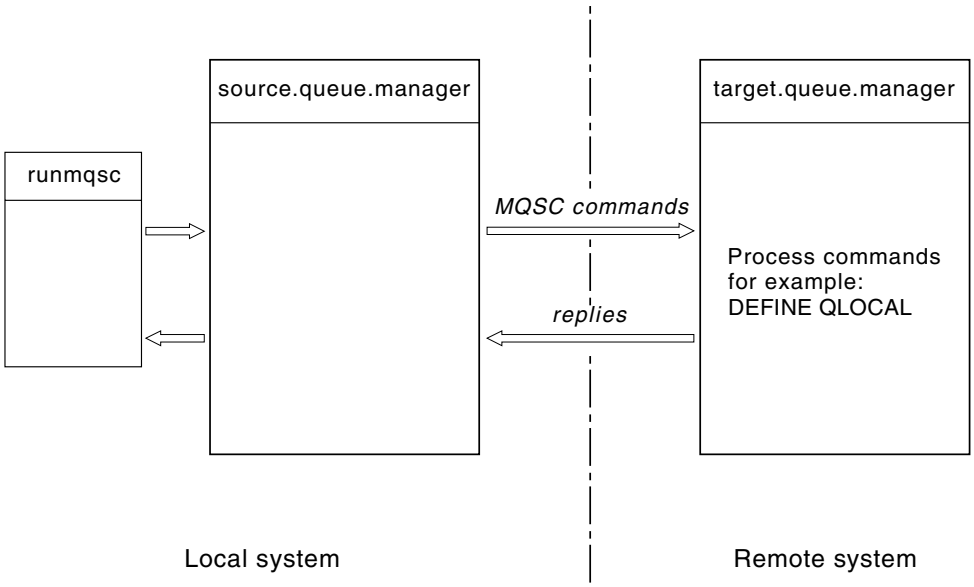


Figure 7. Remote administration using MQSC commands

On both systems, if you have not already done so:

- Create the queue manager and the default objects, using the **crtmqm** command.
- Start the queue manager, using the **strmqm** command.

On the target queue manager:
- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This queue is created by default when a queue manager is created.
- Start the command server, using the **strmqcsv** command.

You have to run these commands locally or over a network facility such as Telnet.

Preparing channels and transmission queues for remote administration

To run MQSC commands remotely, set up two channels, one for each direction, and their associated transmission queues. This example assumes that you are using TCP/IP as the transport type and that you know the TCP/IP address involved.

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the target queue manager. Its sender is at `source.queue.manager` and its receiver is at `target.queue.manager`. The channel `target.to.source` is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each channel. This queue is a local queue that is given the name of the receiving queue manager. The XMITQ name must match the remote queue manager name in order for remote administration to work, unless you are using a queue manager alias. Figure 8 summarizes this configuration.

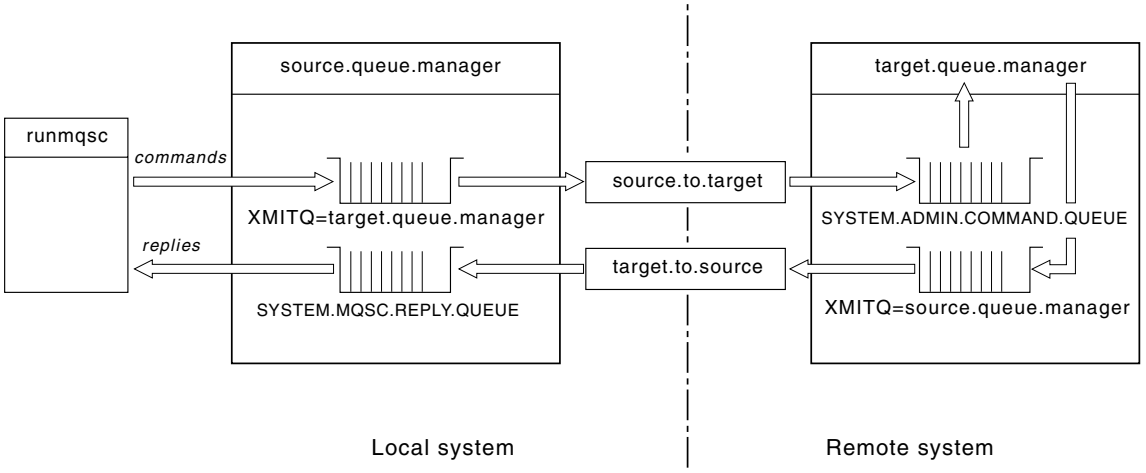


Figure 8. Setting up channels and queues for remote administration

See *WebSphere MQ Intercommunication* for more information about setting up channels.

Defining channels and transmission queues

On the source queue manager, issue the following MQSC commands to define the channels and the transmission queue:

1. Define the sender channel at the source queue manager:

```
DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(SDR) +
       CONNAME (RHX5498) +
       XMITQ ('target.queue.manager') +
       TRPTYPE(TCP)
```

2. Define the receiver channel at the source queue manager:

```
DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)
```

3. Define the transmission queue on the source queue manager:

```
DEFINE QLOCAL ('target.queue.manager') +
       USAGE (XMITQ)
```

Issue the following commands on the target queue manager (`target.queue.manager`), to create the channels and the transmission queue there:

1. Define the sender channel on the target queue manager:

```
DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(SDR) +
       CONNAME (RHX7721) +
       XMITQ ('source.queue.manager') +
       TRPTYPE(TCP)
```

2. Define the receiver channel on the target queue manager:

```
DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)
```

3. Define the transmission queue on the target queue manager:

```
DEFINE QLOCAL ('source.queue.manager') +
       USAGE (XMITQ)
```

Note: The TCP/IP connection names specified for the CONNAME attribute in the sender channel definitions are for illustration only. This is the network name of the machine at the *other* end of the connection. Use the values appropriate for your network.

Starting the channels

The way in which you start the channels depends on the environments in which WebSphere MQ is running.

In Windows systems, start a listener as a background process at the receiver end of each channel.

- On the source queue manager, type:

```
runmqlsr -t TCP -m source.queue.manager
```

- On the target queue manager, type:

```
runmqlsr -t TCP -m target.queue.manager
```

Then start the channels, again as background processes:

- On the source queue manager, type:

```
runmqchl -c source.to.target -m source.queue.manager
```

- On the target queue manager, type:

```
runmqchl -c target.to.source -m target.queue.manager
```

On WebSphere MQ for UNIX systems, start a listener at the receiver end of each channel:

- On the source queue manager, type:

```
runmqlsr -t TCP -m source.queue.manager
```

- On the target queue manager, type:

```
runmqlsr -t TCP -m target.queue.manager
```

Then start the channels as background processes:

- On the source queue manager, type:

  ```
  runmqchl -c source.to.target -m source.queue.manager &
  ```

- On the target queue manager, type:

  ```
  runmqchl -c target.to.source -m source.queue.manager &
  ```

The **runmqlsr** and **runmqchl** commands are WebSphere MQ control commands. You cannot issue them using **runmqsc**. You can, however, start channels using **runmqsc** commands or scripts (start channel).

Automatic definition of channels: If WebSphere MQ receives an inbound attach request and cannot find an appropriate receiver or server-connection definition in the channel definition file (CDF), it creates a definition automatically and adds it to the CDF. Automatic definitions are based on two default definitions supplied with WebSphere MQ: SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN.

You enable automatic definition of receiver and server-connection definitions by updating the queue manager object using the MQSC command, ALTER QMGR (or the PCF command Change Queue Manager).

For more information about creating channel definitions automatically, see *WebSphere MQ Intercommunication*. For information about automatically defining channels for clusters, see *WebSphere MQ Queue Manager Clusters*.

Managing the command server for remote administration

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command.

A command server is mandatory for all administration involving PCF commands, the MQAI, and also for remote administration.

Note: For remote administration, ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager. Avoid this situation.

There are separate control commands for starting and stopping the command server. Users of WebSphere MQ for Windows can perform the operations described in the following sections using the WebSphere MQ Services snap-in. For more information, see Chapter 8, “Administration using the WebSphere MQ Services snap-in” on page 81.

Starting the command server

To start the command server, use the command:

```
strmqcsv saturn.queue.manager
```

where `saturn.queue.manager` is the queue manager for which the command server is being started.

Displaying the status of the command server

For remote administration, ensure that the command server on the target queue manager is running. If it is not running, remote commands cannot be processed. Any messages containing commands are queued in the target queue manager's command queue.

To display the status of the command server for a queue manager called `saturn.queue.manager`, the command is:

```
dspmqcsv saturn.queue.manager
```

You must issue this command on the target machine. If the command server is running, the following message is returned:

```
AMQ8027    WebSphere MQ Command Server Status ..: Running
```

Stopping a command server

To end the command server started by the previous example use the following command:

```
endmqcsv saturn.queue.manager
```

You can stop the command server in two ways:

- For a controlled stop, use the **endmqcsv** command with the -c flag, which is the default.
- For an immediate stop, use the **endmqcsv** command with the -i flag.

Note: Stopping a queue manager also ends the command server associated with it.

Issuing MQSC commands on a remote queue manager

The command server ***must*** be running on the target queue manager, if it is going to process MQSC commands remotely. (This is not necessary on the source queue manager.)

- On the target queue manager, type:

  ```
  strmqcsv target.queue.manager
  ```

- On the source queue manager, you can then run MQSC commands interactively in indirect mode by typing:

  ```
  runmqsc -w 30 target.queue.manager
  ```

This form of the **runmqsc** command, with the -w flag, runs the MQSC commands in indirect mode, where commands are put (in a modified form) on the command-server input queue and executed in order.

When you type in an MQSC command, it is redirected to the remote queue manager, in this case, `target.queue.manager`. The timeout is set to 30 seconds; if a reply is not received within 30 seconds, the following message is generated on the local (source) queue manager:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

When you stop issuing MQSC commands, the local queue manager displays any timed-out responses that have arrived and discards any further responses.

In indirect mode, you can also run an MQSC command file on a remote queue manager. For example:

```
runmqsc -w 60 target.queue.manager < mycomds.in > report.out
```

where `mycomds.in` is a file containing MQSC commands and `report.out` is the report file.

Working with queue managers on z/OS

You can issue MQSC commands to a z/OS queue manager from a queue manager on the platforms described in this book. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the -x flag to the **runmqsc** command on the source node to specify that the target queue manager is running under z/OS:

```
runmqsc -w 30 -x target.queue.manager
```

Recommendations for issuing commands remotely

When you are issuing commands on a remote queue manager:

1. Put the MQSC commands to be run on the remote system in a command file.
2. Verify your MQSC commands locally, by specifying the -v flag on the **runmqsc** command.

   You cannot use **runmqsc** to verify MQSC commands on another queue manager.
3. Check that the command file runs locally without error.
4. Run the command file against the remote system.

If you have problems using MQSC commands remotely

If you have difficulty in running MQSC commands remotely, make sure that you have:

- Started the command server on the target queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
  - The channel along which the commands are being sent.
  - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.
- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.
- Sent requests from a source queue manager that do not make sense to the target queue manager (for example, requests that include parameters that are not supported on the remote queue manager).

See also “Resolving problems with MQSC commands” on page 40.

Creating a local definition of a remote queue

A local definition of a remote queue is a definition on a local queue manager that refers to a queue on a remote queue manager.

You do not have to define a remote queue from a local position, but the advantage of doing so is that applications can refer to the remote queue by its locally-defined name instead of having to specify a name that is qualified by the ID of the queue manager on which the remote queue is located.

Understanding how local definitions of remote queues work

An application connects to a local queue manager and then issues an **MQOPEN** call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the target queue, the target queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an **MQPUT** call, specifying the handle returned from the **MQOPEN** call. The queue manager uses the remote queue name and the remote queue manager name in a transmission header at the start of the message. This information is used to route the message to its correct destination in the network.

As administrator, you can control the destination of the message by altering the remote queue definition.

Example

Purpose: An application needs to put a message on a queue owned by a remote queue manager.

How it works: The application connects to a queue manager, for example, `saturn.queue.manager`. The target queue is owned by another queue manager.

On the **MQOPEN** call, the application specifies these fields:

Field value	Description
ObjectName CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the target queue and the target queue manager.
ObjectType (Queue)	Identifies this object as a queue.
ObjectQmgrName Blank or `saturn.queue.manager`	This field is optional. If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition exists.)

After this, the application issues an **MQPUT** call to put a message onto this queue.

On the local queue manager, you can create a local definition of a remote queue using the following MQSC commands:

```
DEFINE QREMOTE (CYAN.REMOTE.QUEUE) +
       DESCR ('Queue for auto insurance requests from the branches') +
       RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE) +
       RQMNAME (jupiter.queue.manager) +
       XMITQ (INQUOTE.XMIT.QUEUE)
```

where:

QREMOTE (CYAN.REMOTE.QUEUE)
Specifies the local name of the remote queue object. This is the name that applications connected to this queue manager must specify in the **MQOPEN** call to open the queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE on the remote queue manager `jupiter.queue.manager`.

Local definition of remote queue

DESCR ('Queue for auto insurance requests from the branches')
Provides additional text that describes the use of the queue.

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)
Specifies the name of the target queue on the remote queue manager. This is the real target queue for messages sent by applications that specify the queue name CYAN.REMOTE.QUEUE. The queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE must be defined as a local queue on the remote queue manager.

RQMNAME (jupiter.queue.manager)
Specifies the name of the remote queue manager that owns the target queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE.

XMITQ (INQUOTE.XMIT.QUEUE)
Specifies the name of the transmission queue. This is optional; if the name of a transmission queue is not specified, a queue with the same name as the remote queue manager is used.

In either case, the appropriate transmission queue must be defined as a local queue with a *Usage* attribute specifying that it is a transmission queue (USAGE(XMITQ) in MQSC commands).

An alternative way of putting messages on a remote queue

Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, including the remote queue manager name, as part of the **MQOPEN** call. In this case, you do not need a local definition of a remote queue. However, this means that applications must either know, or have access to, the name of the remote queue manager at run time.

Using other commands with remote queues

You can use MQSC commands to display or alter the attributes of a remote queue object, or you can delete the remote queue object. For example:

- To display the remote queue's attributes:
  ```
  DISPLAY QUEUE (CYAN.REMOTE.QUEUE)
  ```
- To change the remote queue to enable puts. This does not affect the target queue, only applications that specify this remote queue:
  ```
  ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)
  ```
- To delete this remote queue. This does not affect the target queue, only its local definition:
  ```
  DELETE QREMOTE (CYAN.REMOTE.QUEUE)
  ```

Note: When you delete a remote queue, you delete only the local representation of the remote queue. You do not delete the remote queue itself or any messages on it.

Defining a transmission queue

A transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel.

The channel provides a one-way link to the remote queue manager. Messages are queued at the transmission queue until the channel can accept them. When you define a channel, you must specify a transmission queue name at the sending end of the message channel.

The MQSC command attribute USAGE defines whether a queue is a transmission queue or a normal queue.

Default transmission queues

When a queue manager sends messages to a remote queue manager, it identifies the transmission queue using the following sequence:

1. The transmission queue named on the XMITQ attribute of the local definition of a remote queue.
2. A transmission queue with the same name as the target queue manager. (This value is the default value on XMITQ of the local definition of a remote queue.)
3. The transmission queue named on the DEFXMITQ attribute of the local queue manager.

For example, the following MQSC command creates a default transmission queue on `source.queue.manager` for messages going to `target.queue.manager`:

```
DEFINE QLOCAL ('target.queue.manager') +
       DESCR ('Default transmission queue for target qm') +
       USAGE (XMITQ)
```

Applications can put messages directly on a transmission queue, or indirectly through a remote queue definition. See also “Creating a local definition of a remote queue” on page 66.

Using remote queue definitions as aliases

In addition to locating a queue on another queue manager, you can also use a local definition of a remote queue for both:

- Queue manager aliases
- Reply-to queue aliases

Both types of alias are resolved through the local definition of a remote queue.

You must set up the appropriate channels for the message to arrive at its destination.

Queue manager aliases

An alias is the process by which the name of the target queue manager, as specified in a message, is modified by a queue manager on the message route. Queue manager aliases are important because you can use them to control the destination of messages within a network of queue managers.

You do this by altering the remote queue definition on the queue manager at the point of control. The sending application is not aware that the queue manager name specified is an alias.

For more information about queue manager aliases, see *WebSphere MQ Intercommunication*.

Reply-to queue aliases

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue.

If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

Aliases

A reply-to queue alias is the process by which a reply-to queue, as specified in a request message, is altered by a queue manager on the message route. The sending application is not aware that the reply-to queue name specified is an alias.

A reply-to queue alias lets you alter the name of the reply-to queue and optionally its queue manager. This in turn lets you control which route is used for reply messages.

For more information about request messages, reply messages, and reply-to queues, see the *WebSphere MQ Application Programming Guide*.

For more information about reply-to queue aliases, see *WebSphere MQ Intercommunication*.

Data conversion

Message data in WebSphere MQ-defined formats (also known as *built-in formats*) can be converted by the queue manager from one coded character set to another, provided that both character sets relate to a single language or a group of similar languages.

For example, conversion between coded character sets with identifiers (CCSIDs) 850 and 500 is supported, because both apply to Western European languages.

For EBCDIC new line (NL) character conversions to ASCII, see "All queue managers" on page 94.

Supported conversions are defined in the *WebSphere MQ Application Programming Reference*.

When a queue manager cannot convert messages in built-in formats

The queue manager cannot automatically convert messages in built-in formats if their CCSIDs represent different national-language groups. For example, conversion between CCSID 850 and CCSID 1025 (which is an EBCDIC coded character set for languages using Cyrillic script) is not supported because many of the characters in one coded character set cannot be represented in the other. If you have a network of queue managers working in different national languages, and data conversion among some of the coded character sets is not supported, you can enable a default conversion. Default data conversion is described in "Default data conversion" on page 71.

File ccsid.tbl

The file ccsid.tbl is used for the following purposes:

- In WebSphere MQ for Windows it records all the supported code sets. In UNIX systems the supported code sets are held internally by the operating system.
- It specifies any additional code sets. To specify additional code sets, you need to edit ccsid.tbl (guidance on how to do this is provided in the file).
- It specifies any default data conversion.

You can update the information recorded in ccsid.tbl; you might want to do this if, for example, a future release of your operating system supports additional coded character sets.

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

In WebSphere MQ for UNIX systems, ccsid.tbl is located in directory `/var/mqm/conv/table`.

Default data conversion

If you set up channels between two machines on which data conversion is not normally supported, you must enable default data conversion for the channels to work.

To enable default data conversion, edit the ccsid.tbl file to specify a default EBCDIC CCSID and a default ASCII CCSID. Instructions on how to do this are included in the file. You must do this on all machines that will be connected using the channels. Restart the queue manager for the change to take effect.

The default data-conversion process is as follows:

- If conversion between the source and target CCSIDs is not supported, but the CCSIDs of the source and target environments are either both EBCDIC or both ASCII, the character data is passed to the target application without conversion.
- If one CCSID represents an ASCII coded character set, and the other represents an EBCDIC coded character set, WebSphere MQ converts the data using the default data-conversion CCSIDs defined in ccsid.tbl.

Note: Try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. If you use only the set of characters that is valid for WebSphere MQ object names (as defined in "Names of WebSphere MQ objects" on page 243) you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

Converting messages in user-defined formats

The queue manager cannot convert messages in user-defined formats from one coded character set to another. If you need to convert data in a user-defined format, you must supply a data-conversion exit for each such format. Do **not** use default CCSIDs to convert character data in user-defined formats. For more information about converting data in user-defined formats and about writing data conversion exits, see the *WebSphere MQ Application Programming Guide*.

Changing the queue manager CCSID

When you have used the CCSID attribute of the ALTER QMGR command to change the CCSID of the queue manager, stop and restart the queue manager to ensure that all running applications, including the command server and channel programs, are stopped and restarted.

This is necessary, because any applications that are running when the queue manager CCSID is changed continue to use the existing CCSID.

Part 3. Using snap-ins to administer WebSphere MQ

Chapter 7. Administration using the WebSphere MQ Explorer

This information applies to WebSphere MQ for Windows only.

WebSphere MQ for Windows provides an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. Appendix E, "Comparing command sets" on page 489 shows you what you can do using the WebSphere MQ Explorer.

The WebSphere MQ Explorer allows you to perform remote administration of your network from a computer running a Windows system by pointing the WebSphere MQ Explorer at the queue managers and clusters you are interested in. The platforms and levels of WebSphere MQ that can be administered using the WebSphere MQ Explorer are described in "Prerequisite software" on page 76.

To configure remote WebSphere MQ queue managers so that WebSphere MQ Explorer can administer them, see "Required definitions for administration" on page 77.

What you can do with the WebSphere MQ Explorer

With the WebSphere MQ Explorer, you can:

- 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.
- 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.
- Add an existing queue manager to a cluster using the *Join Cluster* wizard.
- Manage the authentication information object, used with Secure Sockets Layer (SSL) channel security.

The WebSphere MQ Explorer presents information in a style consistent with that of the Microsoft Management Console (MMC) and the other snap-in applications that the MMC supports.

You perform administration tasks using a series of *Property Sheets* and *Property Pages*. A Property Sheet is a tabbed dialog box made up of a collection of Property Pages. The Property Sheet for an object contains all the attributes relating to that object in a series of fields, some of which you can edit. For each of the WebSphere MQ objects, the attributes are divided into categories which then appear as separate pages within the Property Sheet.

Deciding whether to use the WebSphere MQ Explorer

When deciding whether to use the WebSphere MQ Explorer at your installation, bear the following 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. As a rough guide, if your queue managers have more than 200 queues or 100 channels, consider using a third-party enterprise console product instead of the WebSphere MQ Explorer.

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 view can become cumbersome for large clusters. 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.

Large messages
The message browser displays the first 200 messages on a queue. Only the first 1000 bytes of message data contained in a message are formatted and displayed on your screen. Messages containing more than 1000 bytes of message data are not displayed in their entirety.

Repository on z/OS
The WebSphere MQ Explorer cannot administer a cluster whose repository queue managers are on WebSphere MQ for z/OS. To avoid this, nominate an additional repository queue manager on a system that the WebSphere MQ Explorer can administer. By connecting the cluster through this new repository queue manager, you can administer the queue managers in the cluster, subject to the WebSphere MQ Explorer's restrictions for supported levels of WebSphere MQ.

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 the following installed on your computer:

- The Microsoft Management Console Version 1.1 or higher (installed as part of WebSphere MQ for Windows installation)
- Internet Explorer Version 4.01 (SP1) or later (installed as part of WebSphere MQ for Windows installation)

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

Table 2 summarizes the platforms and command levels that support the WebSphere MQ Explorer.

Table 2. Platforms and command levels

Platform	Command level
AIX and UNIX variants	Command level 221 and above
Windows systems	Command level 201 and above

The WebSphere MQ Explorer handles the differences in the capabilities between the different command levels and platforms. However, if it encounters a value that it does not recognize as an attribute for an object, you cannot change the value of that attribute.

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 for any queue manager being administered.
2. A suitable TCP/IP listener exists for every remote queue manager. This can be the WebSphere MQ listener or the inetd daemon as appropriate.
3. The server-connection channel, called SYSTEM.ADMIN.SVRCONN, exists on every remote queue manager. This channel is mandatory for every remote queue manager being administered. Without it, remote administration is not possible.

   You can create the channel using the following MQSC command:

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

   This command creates a basic channel definition. If you want a more sophisticated definition (to set up security, for example), you need additional parameters.

Cluster membership

The WebSphere MQ Explorer needs to maintain up-to-date administration data about clusters so that it can communicate effectively with them and display correct cluster information when requested. In order to do this, the WebSphere MQ Explorer needs the following information from you:

- 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 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 run the WebSphere MQ Explorer

Before the WebSphere MQ Explorer is enabled, you must:

- Ensure that chosen users have the correct level of authorization. This means being one of the following:
  - A member of the **mqm** group
  - A member of the Administrators group on the machine running the WebSphere MQ Explorer

Group membership at logon time is used for authorization purposes. If you change the membership so that a user can access the WebSphere MQ Explorer, that user must log off and log back on again.

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 a blank. 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.

Using a security exit

A more flexible approach is to install a security exit on the server-connection channel 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 *WebSphere MQ for Windows, V5.3 Quick Beginnings*.

Data conversion

When the connection to a queue manager is established, the queue manager's CCSID is also established. This enables the WebSphere MQ Explorer to perform any character set conversions needed to display the data from remote queue managers correctly.

The tables for converting from the UNICODE CCSID to the queue manager CCSID (and vice versa) must be available to the WebSphere MQ Explorer machine otherwise the WebSphere MQ Explorer cannot communicate with the queue manager.

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.

Using the WebSphere MQ Explorer

This section explains how to use the WebSphere MQ Explorer to show or hide queue managers and clusters, to save or load console files, and to switch off the automatic population facility.

Showing and hiding queue managers and clusters

The WebSphere MQ Explorer can display more than one queue manager at a time. The *Show Queue Manager* dialog box (selectable from the popup menu for the Queue Managers node) allows you to choose whether you display information for a local queue manager or for a queue manager on another (remote) machine. To show a local queue manager, select the *Show a local queue manager* radio button, and choose the appropriate queue manager from a list.

To show a remote queue manager, select the *Show a remote queue manager* radio button and type in the name of the remote queue manager and the connection name in the field provided. The connection name is the IP address, or host name, of the machine you are trying to connect to, with an optional port number. This connection name is used to establish a client connection to the remote queue manager using its SYSTEM.ADMIN.SVRCONN server connection channel.

The *Hide Queue Manager* dialog box (which you select from the popup menu for the Queue Managers node) displays a list of all visible queue managers and allows you to select one to hide from view on the console.

Similar facilities exist for hiding and showing clusters. When you show a cluster in the console, you select a repository queue manager in the cluster as the initial point of connection. Within the cluster, the WebSphere MQ Explorer determines the connection information it needs for the member queue managers.

Saving and loading console files

The WebSphere MQ Explorer can save the contents of a console in a file called a *.MSC file*.

The following information is saved in a .MSC file:

- Details of the queue managers and clusters showing in the console. The names of the queue managers that are members of the visible clusters are not saved.
- The security exit name and the security exit data for client connections to remote queue managers.
- Whether the WebSphere MQ Explorer automatically loads the local queue managers and the clusters of which they are members when the .MSC file is loaded.

- Any non-default configuration of columns visible in each view.
- Filtering options for the objects visible in each view.

You can save different views of the network into each .MSC file.

Switching off the automatic population facility

If you load the WebSphere MQ Explorer into the MMC console using the MMC *Add/Remove Snap-in*, the WebSphere MQ Explorer starts up in its default state.

The default behavior is to automatically determine:

- The names of the queue managers on the local machine and add them into the *Queue Managers* node
- Which clusters the local queue managers are part of and add these clusters to the *Clusters* node

If you do not want this default behavior (perhaps you want to save a console with a particular set of queue managers), switch off automatic population by unchecking the checkbox on the properties page for the top-level WebSphere MQ node and then save the console.

Chapter 8. Administration using the WebSphere MQ Services snap-in

This information applies to WebSphere MQ for Windows only.

The WebSphere MQ Services snap-in runs under the *Microsoft Management Console* (MMC). It allows you to perform more advanced tasks, typically associated with setting up and fine tuning the working environment for WebSphere MQ, either locally or remotely within the Windows system domain. It monitors the operation of WebSphere MQ servers and provides extensive error detection and recovery functions.

The WebSphere MQ Services snap-in is an administration tool for experienced staff who are authorized to make changes to WebSphere MQ objects and services.

What you can do with the WebSphere MQ Services snap-in

All the functions offered by the WebSphere MQ Services snap-in can be used to administer local or remote WebSphere MQ for Windows servers, except for the *Alert monitor* function, which records and notifies you of problems in your WebSphere MQ system. This function can be used on your local system only. See "Using the WebSphere MQ alert monitor application" on page 82 for more information.

With the WebSphere MQ Services snap-in, you can:

- Start or stop a queue manager (on your local machine or on remote Windows systems).
- Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- Create and delete command servers, channel initiators, trigger monitors, and listeners.
- Set any of the services to start up automatically during system startup.
- Set specific services to start up automatically when a queue manager is started.
- Modify some of the properties of queue managers. (Others are modified using WebSphere MQ Explorer.) *This is the equivalent of the mqs.ini and qm.ini files used on WebSphere MQ for UNIX systems*.
- Change the default queue manager.
- Manage secure sockets layer (SSL) certificates, associating certificates with queue managers and configuring and setting up certificate stores.
- Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- Modify the behavior of WebSphere MQ if a particular service fails. You could, for example, try to start the service a certain number of times.
- Start or stop the service trace.

The WebSphere MQ Services snap-in presents information in a style consistent with that of the Microsoft Management Console (MMC) and the other snap-in applications that the MMC supports.

You can also configure and control WebSphere MQ services using the **amqmdain** command. This command is described in "amqmdain (WebSphere MQ services control)" on page 253.

Using the WebSphere MQ Services snap-in

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

Green	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 appears. From this menu, select the `MQSeries Services` option to bring up the MMC. The WebSphere MQ Services snap-in is already loaded and is ready to use.

You can save any changes you make to this console view so that each time you start up the WebSphere MQ Services snap-in from the task bar, it appears as you last saved it.

The first time you start the WebSphere MQ Services snap-in, each queue manager you currently have defined shows the services belonging to that queue manager on the right-hand side of the console window. The WebSphere MQ Services snap-in always contains an up-to-date list of the current set of queue managers. You do not have to add or remove any definitions manually. There are icons for the trace and alert monitor functions in addition to those for the queue managers. The trace and alert monitor functions are special services that do not belong to individual queue managers but to the system as a whole.

You cannot stop the alert monitor application. The trace service, when set to automatic startup, starts before any other services or queue managers.

Using the WebSphere MQ alert monitor application

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 Services snap-in 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

Looking at WebSphere MQ alert monitor information

If the task bar icon indicates that an alert has arisen, double click on the icon to open the *Alert Monitor* display. This dialog shows a tree view, grouped by queue manager, of all the alerts that are currently outstanding. Expand the nodes of the tree to see which services are alerted and look at the following pieces of information relating to the service:

- The date and time of the most recent alert for the service
- The command line that failed

- The error message describing why the service failed

WebSphere MQ Services snap-in recovery facilities

If you have set your queue managers to start automatically during system startup, you can configure the behavior of the WebSphere MQ Services snap-in, in the appropriate property pages, to take one of several actions if one or more queue managers fail:

- Restart the queue managers.
- Execute a program. You might like to set it up to notify you of a failure, either through a pager alert or by e-mail.
- Restart the server.
- Log the fact that a failure has occurred, but take no action.

Security

The WebSphere MQ Services snap-in and the components associated with it use the Microsoft Windows security model. It is this security model that allows or denies access to WebSphere MQ services.

The WebSphere MQ Services snap-in uses *Component Object Model* (COM) and *Distributed Component Object Model (DCOM)* technology to communicate between servers and between processes on a server.

The COM server application, *AMQMSRVN*, is shared between any client processes that need to use the WebSphere MQ Services snap-in components (for example, the WebSphere MQ Services snap-in, the alert monitor task bar, and the WebSphere MQ service).

Because AMQMSRVN must be shared between non-interactive and interactive logon sessions, you must launch it under a special user account. This special user account is called MUSR_MQADMIN. When you install WebSphere MQ and run the Prepare WebSphere MQ Wizard for the first time, it creates a local user account for AMQMSRVN called MUSR_MQADMIN with the required settings and permissions. The password for MUSR_MQADMIN is randomly generated when the account is created, and used to configure the logon environment for AMQMSRVN.

The password is not known outside this *onetime* processing and is stored by the Windows operating system in a secure part of the Registry.

In some network configurations, where user accounts are defined on domain controllers that are using the Windows 2000 operating system, the local user account MUSR_MQADMIN might not have the authority it requires to query the group membership of other domain user accounts. The Prepare WebSphere MQ Wizard identifies whether this is the case by carrying out tests and asking the user questions about the network configuration. If the local user account MUSR_MQADMIN does not have the required authority, the Prepare WebSphere MQ Wizard prompts the user for the account details of a domain user account with particular settings and permissions. The online help for the Prepare WebSphere MQ Wizard contains information about the domain user account required. Once the user has entered valid account details for the domain user account into the Prepare WebSphere MQ Wizard, it configures AMQMSRVN to run

under this account instead of the local user account MUSR_MQADMIN. The account details are held in the secure part of the Registry and cannot be read by users.

When the service is running, AMQMSRVN is launched and remains running for as long as the service is running. A WebSphere MQ administrator who logs onto the server after AMQMSRVN is launched can use the WebSphere MQ Services snap-in to administer queue managers on the server. This connects the WebSphere MQ Services snap-in to the existing AMQMSRVN process. These two actions need different levels of permission before they can work:

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

Changing the user name associated with WebSphere MQ Services

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

To change the user name :

1. Create a new user account (for example NEW_NAME)
2. Use the Prepare WebSphere MQ Wizard to enter the account details of the new user account. Alternatively, use the following command line to set the new account:

   ```
   AMQMSRVN -user <domain\>NEW_NAME -password <password>
   ```

   Where `NEW_NAME` is the new user name you have chosen. This can be qualified by a domain name if required. WebSphere MQ allocates the correct security rights and group membership to the new user account

If for any reason you need to reset the user account back to the default MUSR_MQADMIN account, use the following command:

```
AMQMJPSE -r
```

Controlling access

When you install WebSphere MQ, default access permissions are set up for the AMQMSRVN process. These default access permissions grant access to the process to:

- mqm (local WebSphere MQ administrators group)
- Administrators (local administrators of this machine)

These permissions restrict access to the alert monitor task bar application, the WebSphere MQ Services snap-in, and the WebSphere MQ Explorer snap-in to these users and groups only. Other users trying to access these functions are denied access.

Before you can grant or deny users access to the WebSphere MQ Services snap-in, you must configure the access permissions of the objects involved. Use a tool called *DCOMCNFG.EXE*, shipped with Windows systems, to do this.

Using DCOMCNFG.EXE

To start DCOMCNFG.EXE:

1. Click Start
2. Select **Run**
3. Type **dcomcnfg** in the open input field
4. Click OK

A list of applications is displayed. From this list:

1. Find and highlight the **IBM WebSphere MQ Services** entry.
2. Click Properties. This displays information about the location of the DCOM server (AMQMSRVN.EXE), together with its identity and security properties.
3. Select the **Security** page to view or modify the launch, access, or configuration permissions.
4. Stop the WebSphere MQ service from the Windows Services control panel and restart it for your changes to take effect. (If your changes affect a user who is currently logged on, that user must log off and on again.)

In addition to being able to add to the list of users that are allowed access to a service, you can deny access to specific users and groups. This means that you can grant access to a group of users (by specifying a group name) but deny access to individuals within that group.

Controlling remote access

You can also grant or deny access to users of remote machines using DCOMCNFG.EXE.

You can turn the DCOM server on or off for the entire server using the appropriate setting on the Default Properties page.

User rights granted for MUSR_MQADMIN

The following table lists the user rights granted for MUSR_MQADMIN.

Table 3. User rights granted for MUSR_MQADMIN

Logon as batch job	Enables WebSphere MQ Services COM server to run under this user account
Logon as service	Enables users to set the WebSphere MQ service itself to logon using MUSR_MQADMIN if they require
Shut down the system	Allows the WebSphere MQ Service to restart the server if configured to do so when recovery of a service fails

Changing the password of the AMQMSRVN user account

If AMQMSRVN is running under the local user account MUSR_MQADMIN (or another local user account), you can change the password for the account as follows:

1. Stop the WebSphere MQ service.
2. Close any WebSphere MQ programs that are using the AMQMSRVN COM server (this includes snap-ins, alert monitor, task bar, and so on).
3. Use the *User Manager* to change the MUSR_MQADMIN password in the same way that you would change an individual's password. The User Manager is a Windows NT system management tool that allows system administrators to add, delete, or change users on a WebSphere MQ system.

4. Use DCOMCNFG.EXE to bring up the properties pages for the WebSphere MQ service.
5. Select the Identity Page.
6. Modify the password given for the MUSR_MQADMIN user account.

If AMQMSRVN is running under a domain user account, you can also change the password for the account as follows:

1. Change the password for the domain account on the domain controller. You might need to ask your domain administrator to do this for you.
2. Use the Prepare WebSphere MQ Wizard to enter the account details including the new password.

The user account that AMQMSRVN runs under executes any MQSC commands that are issued by user interface applications, or performed automatically on system startup, shutdown, or service recovery. This user account must therefore have WebSphere MQ administration rights. By default it is added to the local **mqm** group on the server. If this membership is removed, the WebSphere MQ service will not work.

If a security problem arises with the DCOM configuration or with the user account that AMQMSRVN runs under, error messages and descriptions appear in the system event log. One common error is for a user not to have access or launch rights to the server. This error appears in the system log as a DCOM error with the following message description:

```
Access denied attempting to launch a DCOM server.  The server is:
{55B99860-F95E-11d1-ABB6-0004ACF79B59}
```

Part 4. Configuring WebSphere MQ

Chapter 9. Configuring WebSphere MQ

This chapter tells you how to change the behavior of WebSphere MQ or an individual queue manager to suit your installation's needs.

You change WebSphere MQ configuration information by changing the values specified on a set of configuration attributes (or parameters) that govern WebSphere MQ.

How you change this configuration information, and where WebSphere MQ stores your changes, is platform-specific:

- WebSphere MQ for Windows uses the WebSphere MQ Services snap-in to make changes to attribute information within the **Windows Registry**. You can also use amqmdain to set some Registry values, as described in "amqmdain (WebSphere MQ services control)" on page 253.
- Users on all other platforms change attribute values by editing the **WebSphere MQ configuration files**.

This chapter describes:

- "Changing configuration information on Windows systems"
- "Changing configuration information on UNIX systems" on page 90

It then describes:

- The attributes you can use to modify WebSphere MQ configuration (for all queue managers) in "Attributes for changing WebSphere MQ configuration information" on page 94
- The attributes you can use to modify the configuration of an individual queue manager in "Changing queue manager configuration information" on page 100

Changing configuration information on Windows systems

All WebSphere MQ configuration information is stored in the Windows Registry. There is a simple, or close correlation between the contents of the Windows Registry and the WebSphere MQ configuration files.

You edit configuration information from the WebSphere MQ Services snap-in (or by using the **amqmdain** command).

Use the WebSphere MQ Services snap-in to make configuration changes only. **Do not** try to edit the Registry system file directly as this might adversely affect the smooth running of both your WebSphere MQ system and your Windows operating system.

Migrating to the Windows Registry

The Windows Registry is created when you install the operating system. When you migrate to WebSphere MQ for Windows Version 5.3, the information in the configuration files you used in any previous version is automatically stored in the Registry by the WebSphere MQ Services snap-in.

If you want to refer back to these files in the future, back them up before starting the migration process.

Viewing configuration information

You can view a description of the keys used by the Windows Registry from the WebSphere MQ Information Center. You can access the WebSphere MQ Information Center from:

- An icon in the Windows Start menu
- An option in the WebSphere MQ First Steps application

Changing configuration information on UNIX systems

On UNIX platforms, you can change WebSphere MQ configuration attributes within:

- A WebSphere MQ configuration file (**mqs.ini**) to effect changes for WebSphere MQ on the node as a whole. There is one mqs.ini file for each node.
- A queue manager configuration file (**qm.ini**) to effect changes for specific queue managers. There is one qm.ini file for each queue manager on the node.

A configuration file (or ***stanza*** file) contains one or more stanzas, which are groups of lines in the .ini file that together have a common function or define part of a system, such as log functions, channel functions, and installable services.

Because the WebSphere MQ configuration file is used to locate the data associated with queue managers, a nonexistent or incorrect configuration file can cause some or all MQSC commands to fail. Also, applications cannot connect to a queue manager that is not defined in the WebSphere MQ configuration file.

Any changes you make to a configuration file do not take effect until the next time the queue manager is started.

Editing configuration files

Before editing a configuration file, back it up so that you have a copy you can revert to if the need arises.

You can edit configuration files either:

- Automatically, using commands that change the configuration of queue managers on the node
- Manually, using a standard text editor

You can edit the default values in the WebSphere MQ configuration files after installation.

If you set an incorrect value on a configuration file attribute, the value is ignored and an operator message is issued to indicate the problem. (The effect is the same as missing out the attribute entirely.)

When you create a new queue manager:

- Back up the WebSphere MQ configuration file
- Back up the new queue manager configuration file

When do you need to edit a configuration file?

You might need to edit a configuration file if, for example:

- You lose a configuration file. (Recover from backup if you can.)
- You need to move one or more queue managers to a new directory.

- You need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- You are advised to do so by your IBM Support Center.

Configuration file priorities

The attribute values of a configuration file are set according to the following priorities:

- Parameters entered on the command line take precedence over values defined in the configuration files
- Values defined in the qm.ini files take precedence over values defined in the mqs.ini file

The WebSphere MQ configuration file, mqs.ini

The WebSphere MQ configuration file, mqs.ini, contains information relevant to all the queue managers on the node. It is created automatically during installation.

The mqs.ini file for WebSphere MQ for UNIX systems is in the `/var/mqm` directory. It contains:

- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each of them

Figure 9 on page 92 shows an example of a WebSphere MQ configuration file:

```
#********************************************************************#
#* Module Name: mqs.ini                                             *#
#* Type       : WebSphere MQ Configuration File                     *#
#* Function   : Define WebSphere MQ resources for the node          *#
#********************************************************************#
AllQueueManagers:
#********************************************************************#
#* The path to the qmgrs directory, below which queue manager data  *#
#* is stored                                                        *#
#********************************************************************#
DefaultPrefix=/var/mqm

LogDefaults:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=0
   LogDefaultPath=/var/mqm/log

QueueManager:
   Name=saturn.queue.manager
   Prefix=/var/mqm
   Directory=saturn!queue!manager

QueueManager:
   Name=pluto.queue.manager
   Prefix=/var/mqm
   Directory=pluto!queue!manager

DefaultQueueManager:
   Name=saturn.queue.manager
```

Figure 9. Example of a WebSphere MQ configuration file for UNIX systems

Queue manager configuration files, qm.ini

A queue manager configuration file, qm.ini, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. The qm.ini file is automatically created when the queue manager with which it is associated is created.

A qm.ini file is held in the root of the directory tree occupied by the queue manager. For example, the path and the name for a configuration file for a queue manager called QMNAME is:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

The queue manager name can be up to 48 characters in length. However, this does not guarantee that the name is valid or unique. Therefore, a directory name is generated based on the queue manager name. This process is known as *name transformation*. For a description, see "Understanding WebSphere MQ file names" on page 18.

Figure 10 on page 93 shows how groups of attributes might be arranged in a queue manager configuration file in WebSphere MQ for UNIX systems.

```
#********************************************************************#
#* Module Name: qm.ini                                              *#
#* Type       : WebSphere MQ queue manager configuration file       *#
#  Function   : Define the configuration of a single queue manager *#
#********************************************************************#
ExitPath:
   ExitsDefaultPath=/var/mqm/exits

Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQ.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfuno.o [1]
   ComponentDataSize=0

Service:
   Name=NameService
   EntryPoints=5

ServiceComponent:
   Service=NameService
   Name=MQ.DCE.name.service
   Module=/opt/mqm/lib/amqzfa [2]
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=0
   LogPath=/var/mqm/log/saturn!queue!manager/

XAResourceManager:
  Name=DB2 Resource Manager Bank
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB
  XACloseString=
  ThreadOfControl=THREAD


CHANNELS:
  MaxChannels = 20         ; Maximum number of Channels allowed.
  MaxActiveChannels = 100  ; Maximum number of Channels allowed to be
                           ; active at any time.
TCP:                       ; TCP/IP entries.
  KeepAlive = Yes          ; Switch KeepAlive on
```

Figure 10. Example queue manager configuration file for WebSphere MQ for UNIX systems

Notes for Figure 10:

1. /usr/mqm/bin/amqfuno.o on AIX
2. /usr/mqm/lib/amqzfa on AIX

Attributes for changing WebSphere MQ configuration information

On Windows systems, you modify configuration information using the properties pages for WebSphere MQ, accessed from the WebSphere MQ Services snap-in. On UNIX systems, you modify the information by editing the mqs.ini configuration file.

All queue managers

Use the All Queue Managers WebSphere MQ properties page, or the `AllQueueManagers` stanza in the mqs.ini file to specify the following information about all queue managers.

DefaultPrefix=*directory_name*

This attribute specifies the path to the qmgrs directory, within which the queue manager data is kept.

If you change the default prefix for the queue manager, replicate the directory structure that was created at installation time (see Figure 32 on page 482).

In particular, you must create the qmgrs structure. Stop WebSphere MQ before changing the default prefix, and restart WebSphere MQ only after you have moved the structures to the new location and changed the default prefix.

Note: Do not delete the `/var/mqm/qmgrs/@SYSTEM` directory.

As an alternative to changing the default prefix, you can use the environment variable MQSPREFIX to override the `DefaultPrefix` for the **crtmqm** command.

ConvEBCDICNewline=NL_TO_LF|TABLE|ISO

EBCDIC code pages contain a new line (NL) character that is not supported by ASCII code pages (although some ISO variants of ASCII contain an equivalent).

Use the ConvEBCDICNewline attribute to specify how WebSphere MQ is to convert the EBCDIC NL character into ASCII format.

NL_TO_LF

Convert the EBCDIC NL character (X'15') to the ASCII line feed character, LF (X'0A'), for all EBCDIC to ASCII conversions.

NL_TO_LF is the default.

TABLE

Convert the EBCDIC NL character according to the conversion tables used on your platform for all EBCDIC to ASCII conversions.

The effect of this type of conversion might vary from platform to platform and from language to language; even on the same platform, the behavior might vary if you use different CCSIDs.

ISO

Convert:

- ISO CCSIDs using the TABLE method
- All other CCSIDs using the NL_TO_CF method

Possible ISO CCSIDs are shown in Table 4.

Table 4. List of possible ISO CCSIDs

CCSID	Code Set
819	ISO8859-1
912	ISO8859-2
915	ISO8859-5
1089	ISO8859-6
813	ISO8859-7
916	ISO8859-8
920	ISO8859-9
1051	roman8

If the ASCII CCSID is not an ISO subset, ConvEBCDICNewline defaults to NL_TO_LF.

For more information about data conversion, see the *WebSphere MQ Application Programming Guide*.

Client exit path

Use the All Queue Managers WebSphere MQ properties page, or the `ClientExitPath` stanza in the mqs.ini file to specify the default path for location of channel exits on the client.

ExitsDefaultPath=*defaultprefix*
The default prefix for the platform.

Default queue manager

On UNIX systems, use the `DefaultQueueManager` stanza to specify the default queue manager for the node. On Windows systems, change the default queue manager by checking the `Make this the default queue manager` box on the `General` page of the properties for that queue manager.

Name=*default_queue_manager*
The default queue manager processes any commands for which a queue manager name is not explicitly specified. The `DefaultQueueManager` attribute is automatically updated if you create a new default queue manager. If you inadvertently create a new default queue manager and then want to revert to the original, alter the `DefaultQueueManager` attribute manually.

Exit properties

On UNIX systems, use the `ExitProperties` stanza to specify configuration options used by queue manager exit programs. On Windows systems, set these options on the `Exits` page of the properties for each individual queue manager.

CLWLMode=SAFE|FAST
The cluster workload exit, CLWL, allows you to specify which cluster queue in the cluster to open in response to an MQI call (**MQOPEN**, **MQPUT**, and so on). The CLWL exit runs either in FAST mode or SAFE mode depending on the value you specify on the CLWLMode attribute. If you omit the CLWLMode attribute, the cluster workload exit runs in SAFE mode.

SAFE
Run the CLWL exit in a separate process from the queue manager. This is the default.

If a problem arises with the user-written CLWL exit when running in SAFE mode, the following happens:

- The CLWL server process (amqzlwa0) fails.
- The queue manager restarts the CLWL server process.
- The error is reported to you in the error log. If an MQI call is in progress, you receive notification in the form of a return code.

The integrity of the queue manager is preserved.

Note: Running the CLWL exit in a separate process can affect performance.

FAST
Run the cluster exit inline in the queue manager process.

Specifying this option improves performance by avoiding the overheads associated with running in SAFE mode, but does so at the expense of queue manager integrity. You should only run the CLWL exit in FAST mode if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager will fail and you run the risk of the integrity of the queue manager being compromised.

Log defaults for WebSphere MQ

Use the `Default Log Settings` WebSphere MQ properties page on Windows systems, or the `LogDefaults` stanza in the mqs.ini file on UNIX systems, to specify information about log defaults for all queue managers. The log attributes are used as default values when you create a queue manager, but can be overridden if you specify the log attributes on the **crtmqm** command. See "crtmqm (create queue manager)" on page 259 for details of this command.

Once a queue manager has been created, the log attributes for that queue manager are taken from the settings described in "Queue manager logs" on page 102.

The default prefix (specified in the `All Queue Managers` information) and log path specified for the particular queue manager allow the queue manager and its log to be on different physical drives. This is the recommended method, although by default they are on the same drive.

For information about calculating log sizes, see "Calculating the size of the log" on page 200.

Note: The limits given in the following parameter list are limits set by WebSphere MQ. Operating system limits might reduce the maximum possible log size.

LogPrimaryFiles=3 | *2-62*
Primary log files are the log files allocated during creation for future use.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogSecondaryFiles=2 | *1-61*
Secondary log files are the log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

For WebSphere MQ for Windows, the default number of log file pages is 256, giving a log file size of 1 MB. The minimum number of log file pages is 32 and the maximum is 16 384.

For WebSphere MQ for UNIX systems, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 16 384.

LogType=CIRCULAR | LINEAR
The type of log to be used. The default is CIRCULAR.

CIRCULAR
Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See "Circular logging" on page 196 for a fuller explanation of circular logging.

LINEAR
For both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See "Linear logging" on page 197 for a fuller explanation of linear logging.

If you want to change the default, you can either edit the `LogType` attribute, or specify linear logging using the **crtmqm** command. You cannot change the logging method after a queue manager has been created.

LogBufferPages=0 | *0-512*
The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 512. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ V5.3 this is 64 (256 KB).

If you specify a number between 1 and 17, the queue manager defaults to 18 (72 KB). If you specify a number between 18 and 512, the queue manager uses the number specified to set the memory allocated.

The value is examined when the queue manager is created or started, and might be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

LogDefaultPath=*directory_name*
The directory in which the log files for a queue manager reside. The directory resides on a local device to which the queue manager can write and,

preferably, on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is:

- `<DefaultPrefix>\log` for WebSphere MQ for Windows where `<DefaultPrefix>` is the value specified on the `DefaultPrefix` attribute on the `All Queue Managers` WebSphere MQ properties page. This value is set at install time.
- `/var/mqm/log` for WebSphere MQ for UNIX systems

Alternatively, you can specify the name of a directory on the **crtmqm** command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the **crtmqm** command, the value of the `LogDefaultPath` attribute in the mqs.ini file is used.

The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager is created, a `LogPath` value is created in the log attributes in the configuration information, giving the complete directory name for the queue manager's log. This value is used to locate the log when the queue manager is started or deleted.

LogWriteIntegrity=SingleWrite | DoubleWrite | TripleWrite
The method the logger uses to reliably write log records.

SingleWrite
Some hardware guarantees that, if a write operation writes a page and fails for any reason, a subsequent read of the same page into a buffer results in each byte in the buffer being either:

- The same as before the write, or
- The byte that should have been written in the write operation

On this type of hardware (for example, ssa write cache enabled), it is safe for the logger to write log records in a single write.

DoubleWrite
In some cases an additional write is necessary to write log records with more integrity.

TripleWrite
In some cases another additional write is necessary to write log records with complete integrity, but at the cost of performance. This is the default value.

Advanced Configuration and Power Interface (ACPI)

Windows 2000 supports the Advanced Configuration and Power Interface (ACPI) standard. This enables Windows users with ACPI enabled hardware to stop and restart channels when the system enters and resumes from suspend mode. Use the ACPI tab to tell WebSphere MQ how to behave when the system receives a suspend request.

DoDialog=Y | N
Displays the dialog at the time of a suspend request.

DenySuspend=Y | N
Denies the suspend request. This is used if DoDialog=N, or if DoDialog=Y and a dialog cannot be displayed, for example, because your laptop lid is closed.

CheckChannelsRunning=Y | N
Checks whether any channels are running. The outcome can determine the outcome of the other settings.

The following table outlines the effect of each combination of these parameters:

DoDialog	DenySuspend	CheckChannels Running	Action
N	N	N	Accept the suspend request.
N	N	Y	Accept the suspend request.
N	Y	N	Deny the suspend request.
N	Y	Y	If any channels are running deny the suspend request; if not accept the request.
Y	N	N	Display the dialog (see note below; accept the suspend request). This is the default.
Y	N	Y	If no channels are running accept the suspend request; if they are display the dialog (see note below; accept the request).
Y	Y	N	Display the dialog (see note below; deny the suspend request).
Y	Y	Y	If no channels are running accept the suspend request; if they are display the dialog (see note below; deny the request).

Note: In cases where the action is to display the dialog, if the dialog cannot be displayed (for example because your laptop lid is closed), the DenySuspend option is used to determine whether the suspend request is accepted or denied.

API exits

Use the API Exits WebSphere MQ properties page, or the `ApiExitTemplate` and `ApiExitCommon` stanza in the mqs.ini file to identify API exit routines for all queue managers. On Windows systems, you can also use the **amqmdain** command to change the Registry entries for API exits. (To identify API exit routines for individual queue managers, you use the `ApiExitLocal` stanza, as described in "API exits" on page 110.)

For a complete description of the attributes for these stanzas, see "Configuring API exits" on page 421.

Queue managers

On UNIX systems, there is one `QueueManager` stanza for every queue manager. These attributes specify the queue manager name, and the name of the directory containing the files associated with that queue manager. The name of the directory

is based on the queue manager name, but is transformed if the queue manager name is not a valid file name. (See "Understanding WebSphere MQ file names" on page 18 for more information about name transformation.)

On Windows systems, this information is held in the Registry. You cannot use the WebSphere MQ Services snap-in to change it.

Name=*queue_manager_name*
The name of the queue manager.

Prefix=*prefix*
Where the queue manager files are stored. By default, this is the same as the value specified on the DefaultPrefix attribute of the All Queue Managers information.

Directory=*name*
The name of the subdirectory under the `<prefix>\QMGRS` directory where the queue manager files are stored. This name is based on the queue manager name, but can be transformed if there is a duplicate name or if the queue manager name is not a valid file name.

Changing queue manager configuration information

The attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ. On Windows systems, you change the information using the properties pages for the queue manager, accessed from the WebSphere MQ Services snap-in. On UNIX systems, you change the information by editing the qm.ini configuration file.

Installable services

Use the Services queue manager properties page, or the `Service` stanza in the qm.ini file, to specify information about an installable service. There must be one set of service data for every service used.

For each component within a service, you must also specify the name and path of the module containing the code for that component. On UNIX systems, use the `ServiceComponent` stanza for this.

Name=AuthorizationService | NameService
The name of the required service.

AuthorizationService
For WebSphere MQ, the Authorization Service component is known as the Object Authority Manager, or OAM.

- In WebSphere MQ for Windows systems, each queue manager has its own key in the Windows Registry. The equivalents for the `Service` and `ServiceComponent` stanzas for the default authorization component are added to the Windows Registry automatically. Add other `ServiceComponent` stanzas manually.
- In WebSphere MQ for UNIX systems, the `AuthorizationService` stanza and its associated `ServiceComponent` stanza are added automatically when the queue manager is created. Add other `ServiceComponent` stanzas manually.

NameService
You can enable a name service. On Windows systems, use the Services page of the properties for the queue manager to add information about the service. On UNIX systems, add a `NameService` stanza to the qm.ini file.

EntryPoints=*number-of-entries*
The number of entry points defined for the service. This includes the initialization and termination entry points.

SecurityPolicy=Default|NTSIDsRequired (WebSphere MQ for Windows only)
The SecurityPolicy attribute applies only if the service specified is the authorization service, that is, the default OAM. The SecurityPolicy attribute allows you to specify the security policy for each queue manager. The possible values are:

Default
Use the default security policy to take effect. If a Windows security identifier (NT SID) is not passed to the OAM for a particular user ID, an attempt is made to obtain the appropriate SID by searching the relevant security databases.

NTSIDsRequired
Pass an NT SID to the OAM when performing security checks.

See "Windows security identifiers (SIDs)" on page 117 for more information.

For more information about installable services and components, see Part 7, "WebSphere MQ installable services and the API exit" on page 327.

For more information about security services in general, see Chapter 10, "WebSphere MQ security" on page 113.

Service components

You need to specify service component information when you add a new installable service. On Windows systems, use the `Service Component` page, displayed when you add configuration information for a new installable service on the `Services` page of the queue manager properties. On UNIX systems, add a `ServiceComponent` stanza for each Service stanza.

In WebSphere MQ for UNIX systems, the authorization service stanza is present by default, and the associated component, the OAM, is active.

Service=*service_name*
The name of the required service. This must match the value specified on the `Name` attribute of the Service configuration information.

Name=*component_name*
The descriptive name of the service component. This must be unique and contain only characters that are valid for the names of WebSphere MQ objects (for example, queue names). This name occurs in operator messages generated by the service. We recommend that this name begins with a company trademark or similar distinguishing string.

Module=*module_name*
The name of the module to contain the code for this component. This must be a full path name.

ComponentDataSize=*size*
The size, in bytes, of the component data area passed to the component on each call. Specify zero if no component data is required.

For more information about installable services and components, see Part 7, "WebSphere MQ installable services and the API exit" on page 327.

Queue manager logs

Use the `Log queue manager` properties page, or the `Log` stanza in the qm.ini file, to specify information about logging on this queue manager.

By default, these settings are inherited from the settings specified for the default log settings for the queue manager (described in "Log defaults for WebSphere MQ" on page 96). Change these settings only if you want to configure this queue manager in a different way.

For information about calculating log sizes, see "Calculating the size of the log" on page 200.

Note: The limits given in the following parameter list are set by WebSphere MQ. Operating system limits might reduce the maximum possible log size.

LogPrimaryFiles=3 | *2-62*
The log files allocated when the queue manager is created.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

LogSecondaryFiles=2 | *1-61*
The log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

In WebSphere MQ for UNIX systems, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 16 384.

In WebSphere MQ for Windows, the default number of log file pages is 256, giving a log file size of 1 MB. The minimum number of log file pages is 32 and the maximum is 16 384.

Note: The size of the log files specified during queue manager creation cannot be changed for a queue manager.

LogType=CIRCULAR | LINEAR
The type of logging to be used by the queue manager. You cannot change the type of logging to be used once the queue manager has been created. Refer to

the description of the LogType attribute in "Log defaults for WebSphere MQ" on page 96 for information about creating a queue manager with the type of logging you require.

CIRCULAR
Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See "Circular logging" on page 196 for a fuller explanation of circular logging.

LINEAR
For both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See "Linear logging" on page 197 for a fuller explanation of linear logging.

LogBufferPages=0|*0-512*
The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 512. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ V5.3 this is 64 (256 KB).

If you specify a number between 1 and 17, the queue manager defaults to 18 (72 KB). If you specify a number between 18 and 512, the queue manager uses the number specified to set the memory allocated.

The value is examined when the queue manager is created or started, and might be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

LogPath=*directory_name*
The directory in which the log files for a queue manager reside. This must exist on a local device to which the queue manager can write and, preferably, on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is:

- `C:\Program Files\IBM\WebSphere MQ\log` in WebSphere MQ for Windows.
- `/var/mqm/log` in WebSphere MQ for UNIX systems.

You can specify the name of a directory on the **crtmqm** command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the **crtmqm** command, the value of the `LogDefaultPath` attribute is used.

In WebSphere MQ for UNIX systems, user ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the log files are in the default locations supplied with the product.

LogWriteIntegrity=SingleWrite | DoubleWrite | TripleWrite
The method the logger uses to reliably write log records.

SingleWrite
Where a non-volatile write cache is used (for example, ssa write cache enabled), it is safe for the logger to write log records in a single write.

DoubleWrite
In some cases an additional write is necessary to write log records with more integrity.

TripleWrite
In some cases another additional write is necessary to write log records with complete integrity, but at the cost of performance. This is the default value.

Restricted mode

This option applies to UNIX systems only. The `RestrictedMode` stanza is set by the -g option on the **crtmqm** command. Do **not** change this stanza after the queue manager has been created. If you do not use the -g option, the stanza is not created in the qm.ini file.

ApplicationGroup
The name of the group with members that are allowed to:
- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories

XA resource managers

Use the `Resources` queue manager properties page, or the `XAResourceManager` stanza in the qm.ini file, to specify the following information about the resource managers involved in global units of work coordinated by the queue manager.

Add XA resource manager configuration information manually for each instance of a resource manager participating in global units of work; no default values are supplied.

See “Database coordination” on page 140 for more information about resource manager attributes.

Name=*name* (mandatory)
This attribute identifies the resource manager instance.

The `Name` value can be up to 31 characters in length. You can use the name of the resource manager as defined in its XA-switch structure. However, if you are using more than one instance of the same resource manager, you must construct a unique name for each instance. You can ensure uniqueness by including the name of the database in the `Name` string, for example.

WebSphere MQ uses the `Name` value in messages and in output from the **dspmqtrn** command.

Do not change the name of a resource manager instance, or delete its entry from the configuration information, once the associated queue manager has started and the resource manager name is in effect.

SwitchFile=*name* (mandatory)
The fully-qualified name of the load file containing the resource manager’s XA switch structure.

XAOpenString=*string* **(optional)**
The string of data to be passed to the resource manager's xa_open entry point. The contents of the string depend on the resource manager itself. For example, the string could identify the database that this instance of the resource manager is to access. For more information about defining this attribute, see:

- "Adding resource manager configuration information for DB2" on page 148
- "Adding resource manager configuration information for Oracle" on page 150
- "Adding resource manager configuration information for Sybase" on page 152

and consult your resource manager documentation for the appropriate string.

XACloseString=*string* **(optional)**
The string of data to be passed to the resource manager's xa_close entry point. The contents of the string depend on the resource manager itself. For more information about defining this attribute, see:

- "Adding resource manager configuration information for DB2" on page 148
- "Adding resource manager configuration information for Oracle" on page 150
- "Adding resource manager configuration information for Sybase" on page 152

and consult your database documentation for the appropriate string.

ThreadOfControl=THREAD | PROCESS
This attribute is mandatory for WebSphere MQ for Windows. The queue manager uses this value for serialization when it needs to call the resource manager from one of its own multithreaded processes.

THREAD
The resource manager is fully *thread aware*. In a multithreaded WebSphere MQ process, XA function calls can be made to the external resource manager from multiple threads at the same time.

PROCESS
The resource manager is not *thread safe*. In a multithreaded WebSphere MQ process, only one XA function call at a time can be made to the resource manager.

The `ThreadOfControl` entry does not apply to XA function calls issued by the queue manager in a multithreaded application process. In general, an application that has concurrent units of work on different threads requires this mode of operation to be supported by each of the resource managers.

Channels

Use the `Channels` queue manager properties page, or the `CHANNELS` stanza in the qm.ini file, to specify information about channels.

MaxChannels=100 | *number*
The maximum number of channels allowed. The default is 100.

MaxActiveChannels=*MaxChannels_value*
The maximum number of channels allowed to be active at any time. The default is the value specified on the `MaxChannels` attribute.

MaxInitiators=3 | *number*
The maximum number of initiators.

MQIBINDTYPE=FASTPATH | STANDARD
The binding for applications:

FASTPATH
Channels connect using **MQCONNX** FASTPATH; there is no agent process.

STANDARD
Channels connect using STANDARD.

PipeLineLength=1 | *number*
The maximum number of concurrent threads a channel will use. The default is 1. Any value greater than 1 is treated as 2.

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

Note: Pipelining is only effective for TCP/IP channels.

AdoptNewMCA=NO | SVR | SDR | RCVR | CLUSRCVR | ALL | FASTPATH
If WebSphere MQ receives a request to start a channel, but finds that an amqcrsta process already exists for the same channel, the existing process must be stopped before the new one can start. The `AdoptNewMCA` attribute allows you to control the end of an existing process and the startup of a new one for a specified channel type.

If you specify the `AdoptNewMCA` attribute for a given channel type, but the new channel fails to start because the channel is already running:

1. The new channel tries to stop the previous one by requesting it to end.
2. If the previous channel server does not respond to this request by the time the AdoptNewMCATimeout wait interval expires, the process (or the thread) for the previous channel server is ended.
3. If the previous channel server has not ended after step 2, and after the AdoptNewMCATimeout wait interval expires for a second time, WebSphere MQ ends the channel with a `CHANNEL IN USE` error.

Specify one or more values, separated by commas or blanks, from the following list:

NO
The `AdoptNewMCA` feature is not required. This is the default.

SVR
Adopt server channels.

SDR
Adopt sender channels.

RCVR
Adopt receiver channels.

CLUSRCVR
Adopt cluster receiver channels.

ALL
Adopt all channel types except FASTPATH channels.

FASTPATH
Adopt the channel if it is a FASTPATH channel. This happens only if the appropriate channel type is also specified, for example, AdoptNewMCA=RCVR,SVR,FASTPATH.

Attention!
The AdoptNewMCA attribute might behave in an unpredictable fashion with FASTPATH channels. Exercise great caution when enabling the AdoptNewMCA attribute for FASTPATH channels.

AdoptNewMCATimeout=60|1 – 3600
The amount of time, in seconds, that the new process waits for the old process to end. Specify a value in the range 1 – 3600. The default value is 60.

AdoptNewMCACheck=QM|ADDRESS|NAME|ALL
The type of checking required when enabling the AdoptNewMCA attribute. If possible, perform all three of the following checks to protect your channels from being shut down, inadvertently or maliciously. At the very least, check that the channel names match.

Specify one or more values, separated by commas or blanks, to tell the listener process to:

QM
Check that the queue manager names match.

ADDRESS
Check the communications address. For example, the TCP/IP address.

NAME
Check that the channel names match.

ALL
Check for matching queue manager names, the communications address, and for matching channel names.

AdoptNewMCACheck=NAME,ADDRESS is the default for FAP1, FAP2, and FAP3, while AdoptNewMCACheck=NAME,ADDRESS,QM is the default for FAP4 and later.

LU62, NETBIOS, TCP, and SPX

Use these queue manager properties pages, or these stanzas in the qm.ini file, to specify network protocol configuration parameters. They override the default attributes for channels.

LU62 (WebSphere MQ for Windows only)

TPName
The TP name to start on the remote site.

Library1=*DLLName 1*
The name of the APPC DLL.

The default value is WCPIC32.

Library2=*DLLName2*
The same as Library1, used if the code is stored in two separate libraries.

The default value is WCPIC32.

NETBIOS (WebSphere MQ for Windows only)

LocalName=*name*
The name by which this machine is known on the LAN.

AdapterNum=0|*adapter_number*
The number of the LAN adapter. The default is adapter 0.

NumSess=1 | *number_of_sessions*
The number of sessions to allocate. The default is 1.

NumCmds=1 | *number_of_commands*
The number of commands to allocate. The default is 1.

NumNames=1 | *number_of_names*
The number of names to allocate. The default is 1.

Library1=*DLLName1*
The name of the NetBIOS DLL.

The default value is NETAPI32.

TCP

Port=1414 | *port_number*
The default port number, in decimal notation, for TCP/IP sessions. The *well known* port number for WebSphere MQ is 1414.

Library1=*DLLName1* **(WebSphere MQ for Windows only)**
The name of the TCP/IP sockets DLL.

The default is WSOCK32.

KeepAlive=YES | **NO**
Switch the KeepAlive function on or off. KeepAlive=YES causes TCP/IP to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

ListenerBacklog=number
Override the default number of outstanding requests for the TCP/IP listener.

When receiving on TCP/IP, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the TCP/IP port for the listener to accept the request. The default listener backlog values are shown in Table 5.

Table 5. Default outstanding connection requests (TCP)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5
Linux	100
Solaris	100
HP-UX	20
AIX V4.2 or later	100
AIX V4.1 or earlier	10

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

If the backlog reaches the values shown in Table 5, the TCP/IP connection is rejected and the channel cannot start. For message channels, this results in the channel going into a RETRY state and retrying the connection at a later time. For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from **MQCONN** and retries the connection at a later time.

SPX (WebSphere MQ for Windows only)

Socket=5E86 | *socket_number*
The SPX socket number in hexadecimal notation. The default is X'5E86'.

BoardNum=0 | *adapter_number*
The LAN adapter number. The default is adapter 0.

KeepAlive=YES | **NO**
Switch the KeepAlive function on or off.

KeepAlive=YES causes SPX to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

LibraryName1=*DLLName1*
The name of the SPX DLL.

The default is WSOCK32.DLL.

LibraryName2=*DLLName2*
The same as LibraryName1, used if the code is stored in two separate libraries.

The default is WSOCK32.DLL.

ListenerBacklog=number
Override the default number of outstanding requests for the SPX listener.

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the SPX socket for the listener to accept the request. The default listener backlog values are shown in Table 6.

Table 6. Default outstanding connection requests (SPX)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

If the backlog reaches the values shown in Table 6, the SPX connection is rejected and the channel cannot start. For message channels, this results in the channel going into a RETRY state and retrying the connection at a later time. For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from **MQCONN** and should retry the connection at a later time.

Exit path

Use the Exits queue manager properties page, or the `ExitPath` stanza in the qm.ini file to specify the path for user exit programs.

ExitDefaultPath=*string*
The ExitDefaultPath attribute specifies the location of:
- Channel exits for clients
- Channel exits and data conversion exits for servers

The exit path for clients is held in the WebSphere MQ configuration information (as described in "Client exit path" on page 95).

API exits

Use the Exits queue manager properties page, or the ApiExitLocal stanza in the qm.ini file to identify API exit routines for a queue manager. On Windows systems, you can also use the amqmdain command to change the Registry entries for API exits. (To identify API exit routines for all queue managers, you use the ApiExitCommon and ApiExitTemplate stanzas, as described in "API exits" on page 99.)

For a complete description of the attributes for these stanzas, see "Configuring API exits" on page 421.

User datagram protocol (UDP)

The UDP stanza can be used to tailor User Datagram Protocol (UDP) support on your WebSphere MQ system and is applicable to WebSphere MQ for AIX only. UDP is part of the Internet suite of protocols and can be used as an alternative to TCP/IP.

You can use UDP to send message data between MQSeries for Windows Version 2.02 systems (that is with CSD 2 installed) and MQSeries for AIX server systems.

A sample qm.ini file is shipped in the MQM\QMGRS\ directory on MQSeries for Windows Version 2.02 . To use it, copy it to the subdirectory for the queue manager and edit it as required, using the following attribute descriptions to guide you.

ACKREQ_TIMEOUT=5|1–30 000
The time, in seconds, that the internal state machines wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1 – 30 000.

ACKREQ_RETRY=60|1–30 000
The number of times that the internal state machines re-send protocol datagrams before giving up and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1 – 30 000.

CONNECT_TIMEOUT=5|1–30 000
The time, in seconds, that the internal state machines wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1 – 30 000.

CONNECT_RETRY=60|1–30 000
The number of times that the internal state machines re-send protocol datagrams before giving up and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1 – 30 000.

ACCEPT_TIMEOUT=5|1–30 000
The time, in seconds, that the internal state machines wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1–30 000.

ACCEPT_RETRY=60|1–30 000
The number of times that the internal state machines re-send protocol datagrams before giving up and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1 – 30 000.

DROP_PACKETS=0|1–30 000
Test the robustness of the protocols against lost datagrams. Changing the value to something other than 0 causes datagrams to be thrown away and the protocol causes them to be sent again. Do not change the value of this attribute to anything other than 0 for normal usage.

BUNCH_SIZE=8|1–30 000
The number of datagrams that are sent before an acknowledgement datagram is sent from the receiving node. The default is 8.

Changing the default to a value higher than 8 might reduce the number of datagrams sent, but might also affect other aspects of performance. Without knowing the details of the network involved, it is difficult to suggest exactly how to vary the values on this attribute, but a good rule of thumb is that the longer the network delay, the larger the value of BUNCH_SIZE should be for optimum performance.

PACKET_SIZE=2048|512–8192
The maximum size of UDP datagrams sent over the IP network. Some networks might have a limit as low as 512 bytes. The default value of 2048 is successful in most circumstances. However, if you experience problems with this value, you can slowly increase it from 512 until you find your own optimum value.

PSEUDO_ACK=NO|YES
Set the PSEUDO_ACK attribute to YES if you want the datagram that is about to be sent to be modified so that it requests the remote end of the link to send an *information* datagram back to indicate that the node can be reached. PSEUDO_ACK=YES must be set at both the remote and local ends of the channel.

The default is NO.

The TRANSPORT stanza

Use the `TRANSPORT` stanza to tailor User Datagram Protocol (UDP) support on your WebSphere MQ system, in conjunction with the UDP stanza above.

RETRY_EXIT=exitname
The name of the library that contains the retry exit. The retry exit allows your application to suspend data being sent on a channel when communication is not possible.

For Windows systems, the retry exit name takes the form `xyz.DLL(myexit)`; for AIX systems, the retry exit name takes the form `xyz(myexit)`.

For more information about the retry exit, see “The retry exit” on page 495.

Queue manager configuration file

Chapter 10. WebSphere MQ security

WebSphere MQ queue managers transfer information that is potentially valuable, so you need to use an authority system to ensure that unauthorized users cannot access your queue managers. Consider the following types of security controls:

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

Who can use WebSphere MQ objects
You can define which users (usually applications) can use MQI calls and PCF commands to do the following:

- Who can connect to a queue manager.
- Who can access objects like queues, namelists, processes, and authentication information objects, and what type of access they have to those objects.
- Who can access WebSphere MQ messages.
- Who can access the context information associated with a message.

Channel security
You need to ensure that channels used to send messages to remote systems can access the required resources. You also need to ensure that channels can only be manipulated by authorized users.

You can use standard operating facilities to grant access to program libraries, MQI link libraries, and commands. However, the directory containing queues and other queue manager data is private to WebSphere MQ; do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Authority to administer WebSphere MQ

WebSphere MQ administrators have authority to perform the following tasks:

- Use commands (including the commands to grant WebSphere MQ authorities for other users)
- Access the snap-ins on WebSphere MQ for Windows

To be a WebSphere MQ administrator, you must be a member of a special group called the *mqm* group (or a member of the Administrators group on Windows systems; see below). The mqm group is created automatically when WebSphere MQ is installed; add further users to the group to allow them to perform administration (including the root user on UNIX systems). All members of this group have access to all resources. This access can be revoked only by removing a user from the mqm group.

On UNIX platforms, a special user ID of mqm is also created, for use by the product only. It must never be available to non-privileged users. All WebSphere MQ objects are owned by user ID mqm.

On Windows systems, members of the Administrators group can also administer any queue manager. You can also create a domain mqm group on the domain controller that contains all privileged user IDs active within the domain, and add it to the local mqm group. Some commands, for example **crtmqm**, manipulate

authorities on WebSphere MQ objects and so need authority to work with these objects (as described below). Members of the mqm group have authority to work with all objects, but there might be circumstances on Windows systems when authority is denied if you have a local user and a domain-authenticated user with the same name. This is described in "Principals and groups" on page 116.

You do not need to be a member of the mqm group to do the following:

- Issue commands from an application program that issues PCF commands, or MQSC commands within an Escape PCF command, unless the commands manipulate channels, channel initiators, or channel listeners. (These commands are described in "Channel security" on page 127).
- Issue MQI calls from an application program (unless you want to use the fastpath bindings on the **MQCONNX** call).
- Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures.
- Use the **dspmqtrc** command to display WebSphere MQ formatted trace output.

Managing the mqm group

Security administrators add users who need to administer WebSphere MQ to the mqm group. This includes the root user on UNIX systems. They might also need to remove users who no longer need this authority. These tasks are described in "Creating and managing groups" on page 119.

If your domain controller runs on Windows 2000, your domain administrator might have to set up a special account for WebSphere MQ to use. This is described in the *WebSphere MQ for Windows, V5.3 Quick Beginnings*.

Authority to work with WebSphere MQ objects

Queue managers, queues, processes, namelists, and authentication information objects are all accessed from applications that use MQI calls or PCF commands. These resources are all protected by WebSphere MQ, and applications need to be given permission to access them. The entity making the request might be a user, an application program that issues an MQI call, or an administration program that issues a PCF command. The identifier of the requester is referred to as the *principal*.

Different groups of principals can be granted different types of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group might be allowed only to browse the queue (**MQGET** with browse option). Similarly, some groups might have put and get authority to a queue, but not be allowed to alter attributes of the queue or delete it.

Some operations are particularly sensitive and should be limited to privileged users. For example:

- Accessing some special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE
- Running programs that use full MQI context options
- Creating and deleting application queues

Full control permission for all objects is automatically given to the user ID that creates the object and to members of the mqm group (and to the members of the local Administrators group on Windows systems).

When security checks are made

The security checks made for a typical application are as follows:

Connecting to the queue manager (MQCONN or MQCONNX calls)
This is the first time that the application is associated with a particular queue manager. The queue manager interrogates the operating environment to discover the user ID associated with the application. WebSphere MQ then verifies that the user ID is authorized to connect to the queue manager and retains the user ID for future checks.

Users do not have to sign on to WebSphere MQ; WebSphere MQ assumes that users have signed on to the underlying operating system and have been authenticated by that.

Opening the object (MQOPEN or MQPUT1 calls)
WebSphere MQ objects are accessed by opening the object and issuing commands against it. All resource checks are performed when the object is opened, rather than when it is actually accessed. This means that the **MQOPEN** request must specify the type of access required (for example, whether the user wants only to browse the object or perform an update like putting messages onto a queue).

WebSphere MQ checks the resource that is named in the **MQOPEN** request. For an alias or remote queue object, the authorization used is that of the object itself, not the queue to which the alias or remote queue resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias. If a remote queue is referred to explicitly with both the queue and queue manager names, the transmission queue associated with the remote queue manager is checked.

The authority to a dynamic queue is based on that of the model queue from which it is derived, but is not necessarily the same. This is described in Note 1 on page 133.

The user ID used by the queue manager for access checks is the user ID obtained from the operating environment of the application connected to the queue manager. A suitably authorized application can issue an **MQOPEN** call specifying an alternative user ID; access control checks are then made on the alternative user ID. This does not change the user ID associated with the application, only that used for access control checks.

Putting and getting messages (MQPUT or MQGET calls)
No access control checks are performed.

Closing the object (MQCLOSE)
No access control checks are performed, unless the **MQCLOSE** will result in a dynamic queue being deleted. In this case, there is a check that the user ID is authorized to delete the queue.

How access control is implemented by WebSphere MQ

WebSphere MQ uses the security services provided by the underlying operating system. An access control interface called the Authorization Service Interface is part of WebSphere MQ. WebSphere MQ supplies an implementation of an access control manager (conforming to the Authorization Service Interface) known as the *Object Authority Manager (OAM)*. This is automatically installed and enabled for each queue manager you create, unless you specify otherwise (as described in

"Preventing security access checks" on page 127). The OAM can be replaced by any user or vendor written component that conforms to the Authorization Service Interface.

The OAM exploits the security features of the underlying operating system, using operating system user and group IDs. Users can access WebSphere MQ objects only if they have the correct authority. "Using the OAM to control access to objects" on page 122 describes how to grant and revoke this authority.

The OAM maintains an access control list (ACL) for each resource that it controls. Authorization data is stored on a local queue called SYSTEM.AUTH.DATA.QUEUE. WebSphere MQ supplies a command to create and maintain access control lists; do not update this queue in any other way. This is described in "Using the OAM to control access to objects" on page 122.

WebSphere MQ passes the OAM a request containing a principal, a resource name, and an access type. The OAM grants or rejects access based on the ACL that it maintains. WebSphere MQ follows the decision of the OAM; if the OAM cannot make a decision, WebSphere MQ does not allow access.

Identifying the user ID

The OAM needs to be able to identify who is requesting access to a particular resource. WebSphere MQ uses the term *principal* to refer to this identifier. The principal is established when the application first connects to the queue manager; it is determined by the queue manager from the user ID associated with the connecting application. (If the application is running in a transaction monitor environment, the connection is through the X/Open XA interface, which does not provide a mechanism for passing user IDs, so WebSphere MQ assumes a default user name.)

On UNIX systems, the authorization routines check the real (logged-in) user ID.

WebSphere MQ propagates the user ID received from the system in the message header (MQMD structure) of each message as identification of the user. This identifier is part of the message context information and is described in "Context authority" on page 118. Applications cannot alter this information unless they have been authorized to change context information.

Principals and groups

Principals can belong to groups. You can grant access to a particular resource to groups rather than to individuals, to reduce the amount of administration required. For example, you might define a group consisting of users who want to run a particular application. Other users can be given access to all the resources they require simply by adding their user ID to the appropriate group. This is described in "Creating and managing groups" on page 119.

A principal can belong to more than one group (its group set) and has the aggregate of all the authorities granted to each group in its group set. These authorities are cached, so any changes you make to the principal's group membership are not recognized until the queue manager is restarted, unless you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

UNIX systems

All ACLs are based on groups. When a user is granted access to a particular resource, the user ID's primary group is included in the ACL, not the individual user ID, and authority is granted to all members of that

group. Because of this, be aware that you could inadvertently change the authority of a principal by changing the authority of another principal in the same group.

All users are nominally assigned to the default user group *nobody* and by default, no authorizations are given to this group. You can change the authorization in the *nobody* group to grant access to WebSphere MQ resources to users without specific authorizations.

Windows systems

ACLs are based on both user IDs and groups. Checks are the same as for UNIX systems except that individual user IDs can appear in the ACL as well. You can have different users on different domains with the same user ID; WebSphere MQ allows user IDs to be qualified by a domain name so that these users can be given different levels of access. Group names always refer to local groups, so you don't need to qualify them with a domain name.

User IDs can contain up to 20 characters, domain names up to 15 characters, and group names up to 64 characters.

The OAM first checks the local security database, then the database of the primary domain, and finally the database of any trusted domains. The first user ID encountered is used by the OAM for checking. Each of these user IDs might have different group memberships on a particular computer.

Some control commands (for example, **crtmqm**) change authorities on WebSphere MQ objects using the Object Authority Manager (OAM). Because the OAM searches the security databases in the order given above to determine the authority rights for a given user ID, the authority determined by the OAM might override the fact that a user ID is a member of the local mqm group. For example, if you issue **crtmqm** from a user ID authenticated by a domain controller that has membership of the local mqm group through a global group, the command fails if the system has a local user of the same name who is not in the local mqm group.

Windows security identifiers (SIDs)

On Windows systems, the security identifier (SID) is used to supplement the user ID. The SID contains information that identifies the full user account details on the Windows security account manager (SAM) database where the user is defined. When a message is created on WebSphere MQ for Windows, WebSphere MQ stores the SID in the message descriptor. When WebSphere MQ for Windows performs authorization checks, it uses the SID to query the full information from the SAM database. (The SAM database in which the user is defined must be accessible for this query to succeed.)

By default, if a Windows SID is not supplied with an authorization request, WebSphere MQ identifies the user based on the user name alone. It does this by searching the security databases in the following order:

1. The local security database
2. The security database of the primary domain
3. The security database of trusted domains

If the user name is not unique, incorrect WebSphere MQ authority might be granted. To prevent this problem, include an SID in each authorization request; the SID is used by WebSphere MQ to establish user credentials.

Administration authority

To specify that all authorization requests must include an SID, use the Services page of the properties for the queue manager (accessed from the WebSphere MQ Services snap-in). Set the SecurityPolicy to NTSIDsRequired.

Alternate-user authority

You can specify that a user ID can use the authority of another user when accessing a WebSphere MQ object. This is called *alternate-user authority*, and you can use it on any WebSphere MQ object.

Alternate-user authority is essential where a server receives requests from a program and wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example, assume that a server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1. When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message. Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify a different user ID, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user ID when it opens the reply-to queue.

The alternate-user ID is specified on the `AlternateUserId` field of the object descriptor.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. The context information comes in two sections:

Identity section
Who the message came from. It consists of the `UserIdentifier`, `AccountingToken`, and `ApplIdentityData` fields.

Origin section
Where the message came from, and when it was put onto the queue. It consists of the `PutApplType`, `PutApplName`, `PutDate`, `PutTime`, and `ApplOriginData` fields.

Applications can specify the context data when either an **MQOPEN** or **MQPUT** call is made. This data might be generated by the application, passed on from another message, or generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application running under an authorized user ID.

A server program can use the `UserIdentifier` to determine the user ID of an alternate user. You use context authorization to control whether the user can specify any of the context options on any **MQOPEN** or **MQPUT1** call.

See the *WebSphere MQ Application Programming Guide* for information about the context options, and the *WebSphere MQ Application Programming Reference* for descriptions of the message descriptor fields relating to context.

Creating and managing groups

This section tells you how to create groups and add users to them. It also describes how to remove a user from a group. Any changes you make to a principal's group membership are not recognized until the queue manager is restarted, unless you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

Windows NT

On Windows NT, use the User Manager to work with groups.

Creating a group

1. From Administrative Tools (Common) on the Start Programs pulldown, select User Manager to display the User Manager panel.
2. From the User pulldown, select New Local Group.
3. Enter a name for the group. You can also add a description to help identify the group.
4. Click Add to display the Add Users and Groups panel.
5. Highlight the name of a user that you want to add to the group and click Add. The name appears on the list of names to add.
6. If you want to add other users to the group, repeat step 5 for each user.
7. When you have finished adding names to the list, click OK.
8. Click OK to create the group.

Adding a user to a group

1. From Administrative Tools (Common) on the Start Programs pulldown, select User Manager to display the User Manager panel.
2. Highlight the name of the group from the list of groups and double click to display the Local Group Properties panel.
3. Click Add to display the Add Users and Groups panel.
4. Highlight the name of the user that you want to add to the group and click Add. The name appears on the list of names to add.
5. If you want to add other users to the group, repeat step 4 for each user.
6. When you have finished adding names to the list, click OK.
7. Click OK to add the names to the group.

Displaying who is in a group

1. From Administrative Tools (Common) on the Start Programs pulldown, select User Manager to display the User Manager panel.
2. Highlight the name of the group from the list of groups and double click to display the Local Group Properties panel, showing a list of the members of the group.

Removing a user from a group

1. From Administrative Tools (Common) on the Start Programs pulldown, select User Manager to display the User Manager panel.
2. Highlight the name of the group from the list of groups and double click to display the Local Group Properties panel.
3. Select the name of the user that you want to remove and click Remove.
4. If you want to remove other users from the group, repeat Step 3.
5. When you have finished removing names from the list, click OK.

HP-UX

On HP-UX, use the System Administration Manager (SAM) to work with groups.

Creating a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Select Add from the Actions pull down to display the Add a New Group panel.
4. Enter the name of the group and select the users that you want to add to the group.
5. Click Apply to create the group.

Adding a user to a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to add to the group and click Add.
5. If you want to add other users to the group, repeat step 4 for each user.
6. When you have finished adding names to the list, click OK.

Displaying who is in a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel, showing a list of the users in the group.

Removing a user from a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to remove from the group and click Remove.
5. If you want to remove other users from the group, repeat step 4 for each user.
6. When you have finished removing names from the list, click OK.

AIX

On AIX, use SMITTY to work with groups.

Creating a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Add a Group and press Enter.
4. Enter the name of the group and the names of any users that you want to add to the group, separated by commas.
5. Press Enter to create the group.

Adding a user to a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Add the names of the users that you want to add to the group, separated by commas.
6. Press Enter to add the names to the group.

Displaying who is in a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.

Removing a user from a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Delete the names of the users that you want to remove from the group.
6. Press Enter to remove the names from the group.

Solaris

On Solaris, use the `/etc/group` file to work with groups.

Creating a group

To create a new group, type the following command:

```
groupadd group-name
```

Where *group-name* is the name of the group.

Adding a user to a group

To add a user to a group, edit the `/etc/group` file.

Find the line defining the group to which you want to add a member, and add the user ID to the list of user IDs in that group.

For example, the entry for a group called mquser, containing members user1, user2, and user3 might look like this:

```
mquser::42428:root,user1,user2,user3
```

Displaying who is in a group

To display who is a member of a group, look at the entry for that group in the `/etc/group` file.

Removing a user from a group

To remove a member from a group, remove the user ID from the entry for that group in the `/etc/group` file.

Linux

On Linux, use the `/etc/group` file to work with groups.

Creating a group

To create a new group, type the following command:

```
groupadd -g group-ID group-name
```

Where *group-ID* is the numeric identifier of the group, and *group-name* is the name of the group.

Adding a user to a group

To add a user to a group, edit the `/etc/group` file.

Find the line defining the group to which you want to add a member, and add the user ID to the list of user IDs in that group.

For example, the entry for a group called mquser, containing members user1, user2, and user3 might look like this:

```
mquser::42428:root,user1,user2,user3
```

Displaying who is in a group

To display who is a member of a group, look at the entry for that group in the `/etc/group` file.

Removing a user from a group

To remove a member from a the group, remove the user ID from the entry for that group in the `/etc/group` file.

Using the OAM to control access to objects

The OAM provides a command interface for granting and revoking authority to WebSphere MQ objects. You must be suitably authorized to use these commands, as described in "Authority to administer WebSphere MQ" on page 113. User IDs that are authorized to administer WebSphere MQ have *super user* authority to the queue manager, which means that you do not have to grant them further permission to issue any MQI requests or commands.

Giving access to a WebSphere MQ object

Use the **setmqaut** command to give users, and groups of users, access to WebSphere MQ objects. For a full definition of the command and its syntax, see "setmqaut (set or reset authority)" on page 302. The queue manager must be running to use this command. When you have changed access for a principal, the changes are reflected immediately by the OAM.

To give users access to an object, you need to specify:

- The name of the queue manager that owns the objects you are working with; if you do not specify the name of a queue manager, the default queue manager is assumed.
- The name and type of the object (to identify the object uniquely). You specify the name as a *profile*; this is either the explicit name of the object, or a generic name, including wildcard characters. For a detailed description of generic profiles, and the use of wildcard characters within them, see "Using OAM generic profiles" on page 123.
- One or more principals and group names to which the authority applies.

  If a user ID contains spaces, enclose it in single quotes when you use this command. On Windows systems, you can qualify a user ID with a domain

name. If the actual user ID contains an @ symbol, replace this with @@ to show that it is part of the user ID, not the delimiter between the user ID and the domain name.

- A list of authorizations. Each item in the list specifies a type of access that is to be granted to that object (or revoked from it). Each authorization in the list is specified as a keyword, prefixed with a plus sign (+) or a minus sign (–). Use a plus sign to add the specified authorization, and a minus sign to remove the authorization. There must be no spaces between the + or – sign and the keyword.

  You can specify any number of authorizations in a single command. For example, the list of authorizations to permit a user or group to put messages on a queue and to browse them, but to revoke access to get messages is:

  ```
  +browse -get +put
  ```

Examples of using the command

The following examples show how to use the **setmqaut** command to grant and revoke permission to use an object:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE
         -g groupa +browse -get +put
```

In this example:

- `saturn.queue.manager` is the queue manager name
- `queue` is the object type
- `RED.LOCAL.QUEUE` is the object name
- `groupa` is the identifier of the group whose authorizations are to change
- `+browse -get +put` is the authorization list for the specified queue
  - `+browse` adds authorization to browse messages on the queue (to issue **MQGET** with the browse option)
  - `-get` removes authorization to get (**MQGET**) messages from the queue
  - `+put` adds authorization to put (**MQPUT**) messages on the queue

The following command revokes put authority on the queue MyQueue from principal fvuser and from groups groupa and groupb. On UNIX systems, this command also revokes put authority for all principals in the same primary group as fvuser.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser
         -g groupa -g groupb -put
```

Using the command with a different authorization service

If you are using your own authorization service instead of the OAM, you can specify the name of this service on the **setmqaut** command to direct the command to this service. You must specify this parameter if you have multiple installable components running at the same time; if you do not, the update is made to the first installable component for the authorization service. By default, this is the supplied OAM.

Using OAM generic profiles

OAM generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **setmqaut** commands against each individual object when it is created. Using generic profiles in the **setmqaut** command enables you to set a generic authority for all objects that fit that profile.

The rest of this section describes the use of generic profiles in more detail:

Using wildcard characters

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the ? wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

? Use the question mark (?) instead of any single character. For example, `AB.?D` would apply to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

* Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` would apply to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it would **not** apply to `ABC.JKL`; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` would apply to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

** Use the double asterisk (**) **once** in a profile name as:

- The entire profile name to match all object names. For example if you use `-t prcs` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

Note: When using wildcard characters on UNIX systems, you **must** enclose the profile name in quotes.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic

character is more specific then a generic character. So, in the example above, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:
1. ?
2. *
3. **

Dumping profile settings

The **dmpmqaut** command enables you to dump the current authorizations associated with a specified profile.

The following examples show the use of dmpmqaut to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

   Note: UNIX users cannot use the `-p` option; they must use `-g groupname` instead.

2. This example dumps all authority records with a profile that matches queue a.b.c.

   ```
   dmpmqaut -m qmgr1 -n a.b.c -t q
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.c
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - -
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   - - - - - - - - - - - - - - - -
   profile:     a.**
   object type: queue
   entity:      group1
   type:        group
   authority:   get
   ```

3. This example dumps all authority records for profile a.b.*, of type queue.

   ```
   dmpmqaut -m qmgr1 -n a.b.* -t q
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

   The resulting dump would look something like this:

```
profile:     q1
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

```
dmpmqaut -m qmX -l
```

   The resulting dump would look something like this:

```
profile: q1, type: queue
profile: q*, type: queue
profile: name.*, type: namelist
profile: pr1, type: process
```

Note: For WebSphere MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

For detailed information on the command, see "dmpmqaut (dump authority)" on page 265.

Displaying access settings

Use the **dspmqaut** command to view the authorizations that a specific principal or group has for a particular object. The queue manager must be running to use this command. When you change access for a principal using **setmqaut**, the changes are reflected immediately by the OAM. The flags have the same meaning as those in the **setmqaut** command. Authorization can be displayed for only one group or principal at a time. See "dspmqaut (display authority)" on page 271 for a formal specification of this command.

For example, the following command displays the authorizations that the group GpAdmin has to a process definition named Annuities on queue manager QueueMan1.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

Changing and revoking access to a WebSphere MQ object

To change the level of access that a user or group has to an object, use the **setmqaut** command. To revoke the access of a particular user that is a member of a group that has authorization, remove the user from the group, as described in "Creating and managing groups" on page 119.

The user ID that creates a WebSphere MQ object is granted full control authorities to that object. If you remove this user ID from the local mqm group (or the Administrators group on Windows systems) these authorities are not revoked. Use the **setmqaut** command to revoke access to an object for the user ID that created it, after removing it from the mqm or Administrators group.

Preventing security access checks

If you decide that you do not want to perform security checks (for example, in a test environment), you can disable the OAM in one of two ways:

- Set the operating system environment variable MQSNOAUT as follows, before you create a queue manager (if you do this, you cannot add an OAM later):

  On Windows systems:

  ```
  SET MQSNOAUT=yes
  ```

  On UNIX systems:

  ```
  export MQSNOAUT=yes
  ```

- Use the WebSphere MQ Services snap-in or edit the queue manager configuration file to remove the service.

Channel security

Message channel agents (MCAs) are WebSphere MQ applications and need access to various WebSphere MQ resources.

- The user ID associated with a sending channel needs access to the queue manager, the transmission queue, the dead-letter queue, and any resources required by channel exits.
- The user ID associated with the receiving channel needs to open the target queues to put messages onto them. This involves the MQI, so access control checks might need to be made. You can specify whether these checks are made against the user ID associated with the MCA (as described below), or the user ID associated with the message (from the MQMD context field).

  The PUTAUT parameter of the channel definition specifies which user ID is used for these checks.

  – If you use the user ID of the MCA, this user ID will already be defined on the local system.
  – If you use the user ID associated with the message, it is likely that this is a user ID from a remote system. This remote system user ID must be recognized by the target system and have the authority to connect to the queue manager, make inquiries, set attributes, and set context options

(+connect, +inq, +set, and +setall). It must also have authority to put messages and set context information (+put and +setall) for the destination and dead-letter queues.

The user ID associated with the MCA depends on the type of MCA.

Caller MCA
These are MCAs that initiate a channel. They can be started as individual processes, as threads of the channel initiator, or as threads of a process pool. The user ID used is that associated with the parent process (the channel initiator), or the process causing the MCA to be started.

Responder MCA
These are MCAs that are started as a result of a request by a caller MCA. They can be started as individual processes, as threads of the listener, or as threads of a process pool. The user ID can be any one of the following (in this order of preference):

1. On APPC, the caller MCA can indicate the user ID to be used for the responder MCA. This is called the network user ID and applies only to channels started as individual processes. This is set using the USERID parameter of the channel definition.
2. If the USERID parameter is not used, the channel definition of the responder MCA can specify the user ID that the MCA is to use. This is set using the MCAUSER parameter of the channel definition.
3. If the user ID has not been set by either of the methods above, the user ID of the process that starts the MCA or the user ID of the parent process (the listener) is used.

Protecting the definitions associated with channels

WebSphere MQ channels, channel initiators, and listeners are not WebSphere MQ objects; access to them is not controlled by the OAM. WebSphere MQ does not allow users or applications to manipulate these objects, unless their user ID is a member of the mqm group. If you have an application that issues the PCF commands listed below, the user ID specified in the message descriptor of the PCF message must be a member of the mqm group on the target queue manager.

- ChangeChannel
- CopyChannel
- CreateChannel
- DeleteChannel
- PingChannel
- ResetChannel
- StartChannel
- StartChannelInitiator
- StartChannelListener
- StopChannel
- ResolveChannel

A user ID must also be a member of the mqm group on the target machine to issue the equivalent MQSC commands through the Escape PCF command or using **runmqsc** in indirect mode.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required for this. However, putting a message directly on a transmission queue requires special authorization; see Table 9 on page 132.

Channel exits

You can use channel exits for added security. A security exit forms a secure connection between two security exit programs, where one program is for the sending message channel agent (MCA), and one is for the receiving MCA.

On WebSphere MQ for Windows, there is a security exit for both the WebSphere MQ client and the WebSphere MQ server. This is a channel exit program that provides authentication for WebSphere MQ channels by using the Security Services Programming Interface (SSPI). The supplied channel exit programs provide either one-way or two-way (mutual) authentication of a partner system when a session is being established. For a particular channel, each exit program has an associated principal. A connection between two exit programs is an association between the two principals.

The exit source code file is called amqsspin.c, and is stored in the `C:\Program files\IBM\WebSphere MQ\tools\c\samples` directory. If you modify the source code, you must recompile the modified source. The source code does not include any provision for tracing or error handling. If you choose to modify and use the source code, add your own tracing and error-handling routines. You compile and link the file in the same way as any other channel exit, except that SSPI headers need to be accessed at compile time, and SSPI libraries need to be accessed at link time. See *WebSphere MQ for Windows, V5.3 Quick Beginnings* for more information.

See *WebSphere MQ Intercommunication* for more information about channel exits.

Protecting channels with SSL

The Secure Sockets Layer (SSL) protocol provides out of the box channel security, with protection against eavesdropping, tampering, and impersonation. WebSphere MQ support for SSL enables you to specify, on the channel definition, that a particular channel uses SSL security. You can also specify details of the kind of security you want, such as the encryption algorithm you want to use.

SSL support in WebSphere MQ uses the queue manager authentication information object and various MQSC commands and queue manager and channel parameters that define the SSL support required in detail.

The following MQSC commands support SSL:

ALTER AUTHINFO
Modifies the attributes of an authentication information object.

DEFINE AUTHINFO
Creates a new authentication information object.

DELETE AUTHINFO
Deletes an authentication information object.

DISPLAY AUTHINFO
Displays the attributes for a specific authentication information object.

The following queue manager parameters support SSL:

SSLCRLNL
Allows access to a certificate revocation list. The SSLCRLNL attribute specifies a namelist. The namelist contains zero or more authentication information objects. Each authentication information object gives access to an LDAP server.

SSLKEYR
Associates a key repository with a queue manager. The SSLKEYR attribute depends on the platform you are using:

- On UNIX systems, the key database is held in a *GSKit* key database. (The IBM Global Security Kit (GSKit) enables you to use SSL security on UNIX systems.)
- On Windows systems, the key database is held in a Microsoft Certificate store file.

SSLCRYP
On UNIX systems only, sets the SSLCryptoHardware queue manager attribute. This attribute is the name of the parameter string that you can use to configure the cryptographic hardware you have on your system.

The following channel parameters support SSL:

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

SSLCIPH
Specifies the encryption strength and function (CipherSpec), for example NULL_MD5 or RC4_MD5_US. The CipherSpec must match at both ends of channel.

SSLPEER
Specifies the distinguished name (unique identifier) of allowed partners.

This book describes changes to the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands to support the authentication information object. It also describes the **amqmcert** command for managing certificates on Windows systems, and the IKEYCMD command for managing certificates on UNIX systems. See the following sections:

- "setmqaut (set or reset authority)" on page 302
- "dspmqaut (display authority)" on page 271
- "dmpmqaut (dump authority)" on page 265
- "rcrmqobj (recreate object)" on page 288
- "rcdmqimg (record media image)" on page 286
- "dspmqfls (display files)" on page 277
- "amqmcert (manage certificates)" on page 249
- Chapter 18, "Using the IKEYCMD interface to manage keys and certificates on UNIX systems" on page 319

For an overview of channel security using SSL, see *WebSphere MQ Security*.

For details of the new commands and the new channel and queue manager attributes, see the *WebSphere MQ Script (MQSC) Command Reference*.

For details of PCF commands associated with SSL, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

How authorizations work

The authorization specification tables starting on page 132 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:

- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following:

Action to be performed
MQI option, MQSC command, or PCF command.

Access control object
Queue, process, queue manager, namelist, or authentication information object.

Authorization required
Expressed as an MQZAO_ constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **setmqaut** command for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword `+browse`, MQZAO_SET_ALL_CONTEXT corresponds to the keyword `+setall`, and so on. These constants are defined in the header file cmqzc.h, supplied with the product.

Authorizations for MQI calls

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls might require authorization checks: **MQCONN**, **MQOPEN**, **MQPUT1**, and **MQCLOSE**.

For **MQOPEN** and **MQPUT1**, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application might be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of resolving a name that is not a queue manager alias, unless the queue manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the object being opened. In some cases additional queue-independent authority, obtained through an authorization for the queue manager object, is required.

Table 7 on page 132, Table 8 on page 132, Table 9 on page 132, and Table 10 on page 132 summarize the authorizations needed for each call. In the tables *Not applicable* means that authorization checking is not relevant to this operation; *No check* means that no authorization checking is performed.

Note: You will find no mention of namelists or authentication information objects in these tables. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

Authorization specification tables

The special authorization MQZAO_ALL_MQI includes all the authorizations in the tables that are relevant to the object type, except MQZAO_DELETE and MQZAO_DISPLAY, which are classed as administration authorizations.

Table 7. Security authorization needed for MQCONN calls

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

Table 8. Security authorization needed for MQOPEN calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQOO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(2)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(3)**	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(4)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT **(4, 5)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT **(4, 5)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(6)**
MQOO_SET_ ALL_CONTEXT **(4, 7)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6)**
MQOO_OUTPUT (Transmission queue) **(8)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6)**
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	**(9)**	**(9)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(9, 10)**

Table 9. Security authorization needed for MQPUT1 calls

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

Table 10. Security authorization needed for MQCLOSE calls

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

Notes for the tables:

1. If opening a model queue:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
3. This check is performed for all output cases, except transmission queues (see note 8).
4. MQOO_OUTPUT must also be specified.
5. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
6. This authority is required for both the queue manager object and the particular queue.
7. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
8. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
9. At least one of MQOO_INQUIRE (for any object type), or MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET (for queues) must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
10. This authorization allows any *AlternateUserId* to be specified.
11. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
12. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
13. The check is carried out only if both of the following are true:
    - A permanent dynamic queue is being closed and deleted.
    - The queue was not created by the **MQOPEN** call that returned the object handle being used.

    Otherwise, there is no check.

Authorizations for MQSC commands in escape PCFs

Table 11 on page 134 summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that authorization checking is not relevant to this operation.

Authorization specification tables

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC command within the text of the Escape PCF command

Table 11. MQSC commands and security authorization needed

Authorization required for:	Queue object	Process object	Queue manager object	Namelist object	Authentication information object
ALTER *object*	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
CLEAR QLOCAL	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable	Not applicable
DEFINE *object* NOREPLACE **(1)**	MQZAO_CREATE **(2)**	MQZAO_CREATE **(2)**	Not applicable	MQZAO_CREATE **(2)**	MQZAO_CREATE **(2)**
DEFINE *object* REPLACE **(1, 3)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE	MQZAO_CHANGE
DELETE *object*	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE	MQZAO_DELETE
DISPLAY *object*	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY

Notes for Table 11:

1. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **setmqaut** command.
3. This applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

Authorizations for PCF commands

Table 12 summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes all the authorizations in Table 12 that are relevant to the object type, except MQZAO_CREATE, which is not specific to a particular object or object type

Table 12. PCF commands and security authorization needed

Authorization required for:	Queue object	Process object	Queue manager object	Namelist object	Authentication information object
Change *object*	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
Clear Queue	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable	Not applicable
Copy *object* (without replace) **(1)**	MQZAO_CREATE **(2)**	MQZAO_CREATE **(2)**	Not applicable	MQZAO_CREATE (2)	MQZAO_CREATE (2)

Table 12. PCF commands and security authorization needed (continued)

Copy *object* (with replace) **(1, 4)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE	MQZAO_CHANGE
Create *object* (without replace) **(3)**	MQZAO_CREATE **(2)**	MQZAO_CREATE **(2)**	Not applicable	MQZAO_CREATE **(2)**	MQZAO_CREATE **(2)**
Create *object* (with replace) **(3, 4)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE	MQZAO_CHANGE
Delete *object*	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE	MQZAO_DELETE
Inquire *object*	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY
Inquire *object* names	No check	No check	No check	No check	No check
Reset queue statistics	MQZAO_DISPLAY and MQZAO_CHANGE	Not applicable	Not applicable	Not applicable	Not applicable

Notes for Table 12:

1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **setmqaut** command.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
4. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

Guidelines for Windows 2000

This information applies only when you are running WebSphere MQ in a Windows 2000 environment.

WebSphere MQ for Windows runs on Windows 2000, Windows XP, and Windows NT, but the operation of WebSphere MQ security can be affected by differences between the platforms.

WebSphere MQ security relies on calls to the operating system API for information about user authorizations and group memberships. Some functions do not behave identically on the Windows systems. This section includes descriptions of how those differences might affect WebSphere MQ security when you are running WebSphere MQ in a Windows 2000 environment.

When you get a 'group not found' error

This problem can arise because WebSphere MQ loses access to the local mqm group when Windows 2000 servers are promoted to, or demoted from, domain controllers. The symptom is an error indicating the lack of a local mqm group, for example:

```
>crtmqm qm0
AMQ8066:Local mqm group not found.
```

Altering the state of a machine between server and domain controller can affect the operation of WebSphere MQ, because WebSphere MQ uses a locally-defined mqm group. When a server is promoted to be a domain controller, the scope changes from local to domain local. When the machine is demoted to server, all domain

local groups are removed. This means that changing a machine from server to domain controller and back to server loses access to a local mqm group.

To remedy this problem, re-create the local mqm group using the standard Windows 2000 management tools. Because all group membership information is lost, you must reinstate privileged WebSphere MQ users in the newly-created local mqm group. If the machine is a domain member, you must also add the domain mqm group to the local mqm group to grant privileged domain WebSphere MQ user IDs the required level of authority.

When you have problems with WebSphere MQ and domain controllers

This section describes problems that can arise with security settings when Windows 2000 servers are promoted to domain controllers.

While promoting Windows 2000 servers to domain controllers, you are presented with the option of selecting a default or non-default security setting relating to user and group permissions. This option controls whether arbitrary users are able to retrieve group memberships from the active directory. Because WebSphere MQ relies on group membership information to implement its security policy, it is important that the user ID that is performing WebSphere MQ operations can determine the group memberships of other users.

When a domain is created using the default security option, the default user ID created by WebSphere MQ during the installation process (MUSR_MQADMIN) can obtain group memberships for other users as required. The product then installs normally, creating default objects, and the queue manager can determine the access authority of local and domain users if required.

When a domain is created using the non-default security option, the user ID created by WebSphere MQ during the installation (MUSR_MQADMIN) cannot always determine the required group memberships. In this case, you need to know:

- How Windows 2000 with non-default security permissions behaves
- How to allow domain mqm group members to read group membership
- How to configure WebSphere MQ Services to run under a domain user

Windows 2000 domain with non-default security permissions

If a **local** user installs WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user (MUSR_MQADMIN) created for the WebSphere MQ services (AMQMSRVN) can retrieve the group membership information of the installing user. The Prepare WebSphere MQ Wizard asks the user questions about the network configuration to determine whether there are other user accounts defined on domain controllers running Windows 2000. If so, the WebSphere MQ services need to run under a domain user account with particular settings and authorities. The Prepare WebSphere MQ Wizard prompts the user for the account details of this user. Its online help provides details of the domain user account required that can be sent to the domain administrator.

If a **domain** user installs WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user (MUSR_MQADMIN) created for the WebSphere MQ services (AMQMSRVN) cannot retrieve the group membership information of the installing user. In this case, the Prepare WebSphere MQ Wizard always prompts the user for the account details of the domain user account for the WebSphere MQ services to use.

When WebSphere MQ services needs to use a domain user account, WebSphere MQ cannot operate correctly until this has been configured using the Prepare WebSphere MQ Wizard. This configuration includes creating default objects such as the Default Configuration. The Prepare WebSphere MQ Wizard does not allow the user to continue with other tasks, such as creating the Default Configuration, until the WebSphere MQ services have been configured with a suitable account.

See the *WebSphere MQ for Windows, V5.3 Quick Beginnings* for more information.

Allowing domain mqm group members to read group membership

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

In some situations, you might prefer to change the domain security settings instead, to allow domain **mqm** group members to read group membership information for an arbitrary user. To do this, in `Active Directory Users and Computers`, right-click the domain name, for example `mqdev.hursley.ibm.com`, then:

1. Click Delegate Control, then click Next
2. Click Groups and Users:
   a. Click Add.
   b. Highlight Domain mqm and click Add.
3. Click OK.
4. Highlight the Domain mqm selection and click Next.
5. Select the Create a custom task to delegate check box and click Next.
6. Select Only the following objects in the folder and then search under object types for User objects (the list is alphabetical, so go to the last one).
7. Select User Objects and click Next.
8. Select Property-specific and then search down to:

   ```
   Read Group Membership
   Read groupMembershipSAM
   ```

   (The list is sorted alphabetically on the second word). Select both these check boxes, then click Next.
9. Click Finish.

Configuring WebSphere MQ Services to run under a domain user

Use the Prepare WebSphere MQ Wizard to enter the account details of the domain user account. Alternatively, use the following command line to set the domain user account:

```
AMQMSRVN -user [domain]\[userid] -password [password]
```

In either case, WebSphere MQ allocates the correct security rights and group membership to the new user account.

Applying security template files

Windows 2000 supports text-based security template files that you can use to apply uniform security settings to one or more computers with the Security Configuration and Analysis MMC snap-in. In particular, Windows 2000 supplies

several templates that include a range of security settings with the aim of providing specific levels of security. These include compatible, basic, secure, and highly-secure.

Be aware that applying one of these templates might affect the security settings applied to WebSphere MQ files and directories. If you want to use the highly-secure template, configure your machine before you install WebSphere MQ. If you apply the highly-secure template to a machine on which WebSphere MQ is already installed, all the permissions you have specifically set on the WebSphere MQ files and directories are removed. This means that you lose Administrator and mqm group access and, when applicable, Everyone group access from the error directories.

Nested groups

You can place Windows 2000 domain controllers in *native* mode, which allows users to nest local groups, and also to perform multiple nesting of global and universal groups. The WebSphere MQ security model does not support either nested local groups, or multiple nesting of global and universal groups. The supported group model is the same as for Windows NT, except that you can use universal groups instead of global groups. This means that local and domain local groups are supported, as are any immediately nested global or universal groups.

Chapter 11. Transactional support

This chapter introduces transactional support. The work required to enable your applications to use WebSphere MQ in conjunction with a database product spans the areas of application programming and system administration. Use the information here together with Chapter 13, "Committing and backing out units of work" of the *WebSphere MQ Application Programming Guide*.

We start by introducing the units of work that form transactions, then describe the ways in which you enable WebSphere MQ to coordinate transactions with databases.

Introducing units of work

This section introduces the general concepts of commit, backout, syncpoint, and unit of work. If you are familiar with these transaction processing terms, you can skip to "Scenario 1: Queue manager performs the coordination" on page 140, where we start to deal in more detail with their use with WebSphere MQ.

When a program puts messages on queues within a unit of work, those messages are made visible to other programs only when the program *commits* the unit of work. To commit a unit of work, all updates must be successful to preserve data integrity.

If the program detects an error and decides not to make the put operation permanent, it can *back out* the unit of work. When a program performs a backout, WebSphere MQ restores the queues by removing the messages that were put on the queues by that unit of work.

Similarly, when a program gets messages from one or more queues within a unit of work, those messages remain on the queues until the program commits the unit of work, but the messages are not available to be retrieved by other programs. The messages are permanently deleted from the queues when the program commits the unit of work. If the program backs out the unit of work, WebSphere MQ restores the queues by making the messages available to be retrieved by other programs.

The decision to commit or back out the changes is taken, in the simplest case, at the end of a task. However, it can be more useful for an application to synchronize data changes at other logical points within a task. These logical points are called syncpoints (or synchronization points) and the period of processing a set of updates between two syncpoints is called a *unit of work*. Several MQGET calls and MQPUT calls can be part of a single unit of work.

With WebSphere MQ, we need to distinguish between *local* and *global* units of work:

Local units of work
: Are those in which the only actions are puts to, and gets from, WebSphere MQ queues, and the coordination of each unit of work is provided within the queue manager using a *single-phase commit* process.

  Use local units of work when the only resources to be updated are the queues managed by a single WebSphere MQ queue manager. Updates are committed using the MQCMIT verb or backed out using MQBACK.

There are no system administration tasks, other than log management, involved in using local units of work. In your applications, just start using the MQPMO_SYNCPOINT and MQGMO_SYNCPOINT options on your MQPUT and MQGET calls with MQCMIT and MQBACK. (For information on log management, see "Managing log files" on page 203.)

Global units of work

Are those in which other resources, such as tables in a relational database, are also updated. When more than one *resource manager* is involved, there is a need for *transaction manager* software that uses a *two-phase commit* process to coordinate the global unit of work.

Use global units of work when you also need to include updates to relational database manager software, such as DB2, Oracle, and Sybase.

We define two scenarios for global units of work:

1. In the first, the queue manager itself acts as the transaction manager. In this scenario, MQI verbs control the global units of work; they are started in applications using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK.
2. In the second, the transaction manager role is performed by other software, such as TXSeries™, Encina®, or Tuxedo. In this scenario, an API provided by the transaction manager software is used to control the unit of work (for example, EXEC CICS SYNCPOINT for TXSeries).

The following sections describe all the steps necessary to use global units of work, organized by the two scenarios:

- "Scenario 1: Queue manager performs the coordination" below
- "Scenario 2: Other software provides the coordination" on page 161

Scenario 1: Queue manager performs the coordination

This section describes this scenario, including:

- "Database coordination"
- "DB2 configuration" on page 147
- "Oracle configuration" on page 149
- "Sybase configuration" on page 151
- "Multiple database configurations" on page 152
- "Security considerations" on page 153
- "Administration tasks" on page 154

Database coordination

When the queue manager coordinates global units of work itself, it becomes possible to integrate database updates within the units of work. That is, a mixed MQI and SQL application can be written, and the MQCMIT and MQBACK verbs can be used to commit or roll back the changes to the queues and databases together.

The queue manager achieves this using the two-phase commit protocol described in *X/Open Distributed Transaction Processing: The XA Specification*. When a unit of work is to be committed, the queue manager first asks each participating database manager whether it is prepared to commit its updates. Only if all the participants, including the queue manager itself, are prepared to commit, are all the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET (include the flag MQGMO_SYNCPOINT in the message options)
MQPUT (include the flag MQPMO_SYNCPOINT in the message options)
SQL INSERT
MQCMIT
```

The purpose of MQBEGIN is to denote the beginning of a global unit of work. The purpose of MQCMIT is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

When the unit of work (also known as a *transaction*) is completed successfully using MQCMIT, all actions taken within that unit of work are made permanent or irreversible. If, for any reason, the unit of work fails, all actions are instead backed out. It is not acceptable for one action comprising a unit of work to be made permanent while another is forgotten. This is the principle of a unit of work: either all actions within the unit of work are made permanent or none of them are.

Notes:

1. The application programmer can force a unit of work to be backed out by calling MQBACK. The unit of work is also backed out by the queue manager if the application or database *crashes* before MQCMIT is called.
2. If an application calls MQDISC without calling MQCMIT, the queue manager behaves as if MQCMIT had been called, and commits the unit of work.

In between MQBEGIN and MQCMIT, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the SQL INSERT in the pseudocode above).

Full recovery support is provided if the queue manager loses contact with any of the database managers during the commit protocol. If a database manager becomes unavailable while it is in doubt, that is, it has successfully prepared to commit, but has yet to receive a commit or backout decision, the queue manager remembers the outcome of the unit of work until that outcome has been successfully delivered to the database. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered over queue manager restart. If an application terminates unexpectedly, the integrity of the unit of work is not compromised, but the outcome depends on where in the process the application terminated, as described inTable 14 on page 142.

What happens when the database or application program crashes is summarized in the tables below:

Table 13. What happens when a database server crashes

Before the application call to MQCMIT.	The unit of work is backed out.
During the application call to MQCMIT, **before** all databases have indicated that they have successfully prepared.	The unit of work is backed out with a reason code of MQRC_BACKED_OUT.
During the application call to MQCMIT, **after** all databases have indicated that they have successfully prepared, but before all have indicated that they have successfully committed.	The unit of work is held in recoverable state by the queue manager, with a reason code of MQRC_OUTCOME_PENDING.

Table 13. What happens when a database server crashes (continued)

During the application call to MQCMIT, **after** all databases have indicated that they have successfully committed.	The unit of work is committed with a reason code of MQRC_NONE.
After the application call to MQCMIT.	The unit of work is committed with a reason code of MQRC_NONE.

Table 14. What happens when an application program crashes

Before the application call to MQCMIT.	The unit of work is backed out.
During the application call to MQCMIT, **before** the queue manager has received the application's MQCMIT request.	The unit of work is backed out.
During the application call to MQCMIT, **after** the queue manager has received the application's MQCMIT request.	The queue manager tries to commit using two-phase commit (subject to the database products successfully executing and committing their parts of the unit of work).

In the case where the reason code on return from MQCMIT is MQRC_OUTCOME_PENDING, the unit of work is remembered by the queue manager until it has been able to reestablish contact with the database server, and tell it to commit its part of the unit of work. Refer to "Administration tasks" on page 154 for information on how and when recovery is done.

The queue manager communicates with database managers using the XA interface as described in *X/Open Distributed Transaction Processing: The XA Specification*. Examples of these function calls are xa_open, xa_start, xa_end, xa_prepare, and xa_commit. We use the terms *transaction manager* and *resource manager* in the same sense as they are used in the XA specification.

Restrictions

The following restrictions apply to the database coordination support:

- The ability to coordinate database updates within WebSphere MQ units of work is **not** supported in an MQI client application. The use of MQBEGIN in a client application fails, as described in the *WebSphere MQ Application Programming Reference*. A program that calls MQBEGIN must run as a *server* application on the same machine as the queue manager.

  Note: A *server* application is a program that has been linked with the necessary WebSphere MQ server libraries; a *client* application is a program that has been linked with the necessary WebSphere MQ client libraries. See *WebSphere MQ Clients* and the *WebSphere MQ Application Programming Guide* for details on compiling and linking your programs.
- The database server can reside on a different machine from the queue manager server, as long as the database client is installed on the same machine as the queue manager, and it supports this function. Consult the database product's documentation to determine whether their client software can be used for two-phase commit systems.
- Although the queue manager behaves as a resource manager (for the purposes of being involved in Scenario 2 global units of work), it is not possible to make one queue manager coordinate another queue manager within its Scenario 1 global units of work.

Switch load files

The switch load file is a shared library (a DLL on Windows systems) that is loaded by the code in your WebSphere MQ application and the queue manager. Its purpose is to simplify the loading of the database's client shared library, and to return the pointers to the XA functions.

The path to the switch load file is specified before the queue manager is started. The details are placed in the qm.ini file (UNIX systems), or the Registry (Windows systems). Use the WebSphere MQ Services snap-in to add the details to the Registry, as described in Chapter 8, "Administration using the WebSphere MQ Services snap-in" on page 81.

The C source for the switch load file is supplied with the WebSphere MQ installation if it supports Scenario 1 global units of work. The source contains a function called MQStart. When the switch load file is loaded, the queue manager calls this function, which returns the address of a structure called an *XA switch*.

The XA switch structure exists in the database client shared library, and contains a number of function pointers, as described in Table 15:

Table 15. XA switch function pointers

Function pointer name	XA function	Purpose
xa_open_entry	xa_open	Connect to database
xa_close_entry	xa_close	Disconnect from database
xa_start_entry	xa_start	Start a branch of a global unit of work
xa_end_entry	xa_end	Suspend a branch of a global unit of work
xa_rollback_entry	xa_rollback	Roll back a branch of a global unit of work
xa_prepare_entry	xa_prepare	Prepare to commit a branch of a global unit of work
xa_commit_entry	xa_commit	Commit a branch of a global unit of work
xa_recover_entry	xa_recover	Discover from the database whether it has an in-doubt unit of work
xa_forget_entry	xa_forget	Allow a database to forget a branch of a global unit of work
xa_complete_entry	xa_complete	Complete a branch of a global unit of work

During the first MQBEGIN call in your application, the WebSphere MQ code that executes as part of MQBEGIN loads the switch load file, and calls the xa_open function in the database shared library. Similarly, during queue manager startup, and on other subsequent occasions, some queue manager processes load the switch load file and call xa_open.

You can reduce the number of xa_* calls by using *dynamic registration*. For a complete description of this optimization technique, see "XA dynamic registration" on page 158.

Configuring your system for database coordination

There are several tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager. These are described here as follows:

- "Installing and configuring the database product"
- "Creating switch load files"
- "Adding configuration information to the queue manager" on page 145
- "Writing and modifying your applications" on page 146
- "Testing the system" on page 147

Installing and configuring the database product: The steps involved in installing and configuring your database product are, of course, described in that product's own documentation. Installation issues are well beyond the scope of this chapter, but we can list general configuration issues, as they relate to the interoperation between WebSphere MQ and the database.

Database connections: An application that establishes a standard connection to the queue manager is associated with a thread in a separate local queue manager agent process. (A connection that is not a *fastpath* connection is a *standard* connection in this context. For more information, see "Connecting to a queue manager using the MQCONNX call" in the *WebSphere MQ Application Programming Guide*.)

When the application issues **MQBEGIN**, both it and the agent process call the xa_open function in the database client library. In response to this, the database client library code *connects* to the database that is to be involved in the unit of work *from both the application and queue manager processes*. These database connections are maintained as long as the application remains connected to the queue manager.

This is an important consideration if the database supports only a limited number of users or connections, because two connections are being made to the database to support the one application program.

Client/server configuration: The database client library that is loaded into the WebSphere MQ queue manager and application processes **must** be able to send to and receive from its server. Ensure that:

- The database's client/server configuration files have the correct details
- The relevant environment variables are set in the environment of the queue manager **and** the application processes

Creating switch load files: WebSphere MQ comes with a sample makefile, used to build switch load files for the supported database managers. This makefile, together with all the associated C source files required to build the switch load files, is installed in the following directories:

- For WebSphere MQ for Windows, in the `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm\` directory
- For WebSphere MQ for UNIX systems, in the `/opt/mqm/samp/xatm/` directory (`/usr/mqm/samp/xatm` on AIX)

The sample source modules used to build the switch load files are:

- For DB2, db2swit.c
- For Oracle, oraswit.c
- For Sybase, sybswit.c

Adding configuration information to the queue manager: When you have created a switch load file for your database manager, and placed it in a safe location, you must specify that location to your queue manager. This is done in the queue manager's qm.ini file in the XAResourceManager stanza (UNIX systems), or in the Registry using the WebSphere MQ Services snap-in (Windows systems). Add an XAResourceManager stanza for the database that your queue manager is going to coordinate. The most common case is for there to be only one database, and therefore only one XAResourceManager stanza. More complicated configurations involving multiple databases, are discussed in "Multiple database configurations" on page 152. The attributes of the XAResourceManager stanza are as follows:

Name=name
User-chosen string that identifies the resource manager. In effect, it gives a name to the XAResourceManager stanza. The name is mandatory and can be up to 31 characters in length.

The name you choose must be unique; there must be only one XAResourceManager stanza with this name in this qm.ini file. The name should also be meaningful, because the queue manager uses it to refer to this resource manager both in queue manager error log messages and in output when the **dspmqtrn** command is used. (See "Displaying outstanding units of work with the dspmqtrn command" on page 155 for more information.)

Once you have chosen a name, and have started the queue manager, do not change the Name attribute. This is discussed in more detail in "Changing configuration information" on page 157.

SwitchFile=name
This is the fully-qualified pathname of the XA switch load file you built earlier. This is a mandatory attribute. The code in the queue manager and WebSphere MQ application processes tries to load the switch load file from this pathname on two occasions:

1. At queue manager startup
2. When you make the first call to MQBEGIN in your WebSphere MQ application process

The security and permissions attributes of your switch load file must allow these processes to perform this action.

XAOpenString=string
This is a string of data that WebSphere MQ code passes in its calls to the database manager's xa_open function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and WebSphere MQ application processes call the xa_open function on two occasions:

1. At queue manager startup
2. When you make the first call to MQBEGIN in your WebSphere MQ application process

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the xa_open string contains authentication information (user name and password) to allow a connection to the database in both the queue manager and the application processes.

XACloseString=string
This is a string of data that WebSphere MQ code passes in its calls to the

database manager's xa_open function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and WebSphere MQ application processes call the xa_close function on two occasions:

1. At queue manager startup
2. When you make a call to MQDISC in your WebSphere MQ application process, having earlier made a call to MQBEGIN

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the string is empty, and it is common to omit the XACloseString attribute from the XAResourceManager stanza.

ThreadOfControl=THREAD | PROCESS

The ThreadOfControl value can be THREAD or PROCESS. The queue manager uses it for serialization purposes. This is an optional attribute; if it is omitted, the value PROCESS is assumed.

If the database client code allows threads to call the XA functions without serialization, the value for ThreadOfControl can be THREAD. The queue manager assumes that it can call the XA functions in the database client shared library from multiple threads at the same time, if necessary.

If the database client code does not allow threads to call its XA functions in this way, the value for ThreadOfControl must be PROCESS. In this case, the queue manager serializes all calls to the database client shared library so that only one call at a time is made from within a particular process. You probably also need to ensure that your application performs similar serialization if it runs with multiple threads.

Note that this issue, of the database product's ability to cope with multi-threaded processes in this way, is an issue for that product's vendor. Consult the database product's documentation for details on whether you can set the ThreadOfControl attribute to THREAD or PROCESS. We recommend that, if you can, you set ThreadOfControl to THREAD. If in doubt, the *safer* option is to set it to PROCESS, although you will lose the potential performance benefits of using THREAD.

Writing and modifying your applications: The sample application programs for Scenario 1 global units of work that are supplied with a WebSphere MQ installation are described in the *WebSphere MQ Application Programming Guide*.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET
MQPUT
SQL INSERT
MQCMIT
```

The purpose of MQBEGIN is to denote the beginning of a global unit of work. The purpose of MQCMIT is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

In between MQBEGIN and MQCMIT, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the SQL INSERT in the pseudocode above).

The role of the queue manager, as far as the database is concerned, is to tell it when a global unit of work has started, when it has ended, and whether the global unit of work should be committed or rolled-back.

As far as your application is concerned, the queue manager performs two roles: a resource manager (where the resources are messages on queues) and the transaction manager for the global unit of work.

We recommend that you start with the supplied sample programs, and work through the various WebSphere MQ and database API calls that are being made in those programs. The API calls concerned are fully documented in the *WebSphere MQ Application Programming Guide*, the *WebSphere MQ Application Programming Reference*, and (in the case of the database's own API) the database's own documentation.

Testing the system: You only know whether your application and system are correctly configured by running them during testing. You can test the system's configuration (the successful communication between queue manager and database) by building and running one of the supplied sample programs.

DB2 configuration

The supported levels of DB2 are defined at:

`http://www.ibm.com/software/ts/mqseries/platforms/supported.html`

You need to do the following:

1. Check the environment variable settings.
2. Create the DB2 switch load file.
3. Add resource manager configuration information.
4. Change DB2 configuration parameters if necessary.

Read this information in conjunction with the general information provided in "Configuring your system for database coordination" on page 144.

Checking the DB2 environment variable settings

Ensure that your DB2 environment variables are set for queue manager processes ***as well as in*** your application processes. In particular, you must always set the DB2INSTANCE environment variable ***before*** you start the queue manager. The DB2INSTANCE environment variable identifies the DB2 instance containing the DB2 databases that are being updated. For example:

- On UNIX systems, use:

  ```
  export DB2INSTANCE=db2inst1
  ```

- On Windows systems, use:

  ```
  set DB2INSTANCE=DB2
  ```

Creating the DB2 switch load file

The easiest way to create the DB2 switch load file is to use the sample file xaswit.mak, which WebSphere MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find xaswit.mak in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the DB2 switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak db2swit.dll
```

On AIX, you can find xaswit.mak in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`. On all UNIX systems the directory also contains the C source files that you need to build the switch load file.

Edit xaswit.mak to *uncomment* the lines appropriate to the version of DB2 you are using. Then execute the makefile using the command:

```
make -f xaswit.mak db2swit
```

The generated switch file is placed in your current directory.

Adding resource manager configuration information for DB2

The next step is to modify the configuration information for the queue manager, as described in "Adding configuration information to the queue manager" on page 145, to declare DB2 as a participant in global units of work. On Windows systems, use the Resources page of the properties for the queue manager (accessed from the WebSphere MQ Services snap-in). On WebSphere MQ for UNIX systems, use the `XAResourceManager` stanza of the qm.ini configuration file.

Figure 11 is a UNIX sample, showing some XAResourceManager entries where the database to be updated is called `mydbname`, this name being specified in the `XAOpenString`.:

```
XAResourceManager:
  Name=mydb2
  SwitchFile=/home/myuser/mylibraries/db2swit
  XAOpenString=mydbname,myuser,mypasswd
  ThreadOfControl=PROCESS
```

Figure 11. Sample XAResourceManager entry for DB2 on UNIX platforms

Note: DB2 Version 7.x introduces an improved XAOpenString format. However, it continues to support the method shown above, so we have used it as the method that applies to all DB2 versions.

Changing DB2 configuration parameters

For each DB2 database that the queue manager is coordinating, you need to:

Set database privileges

The queue manager processes run with effective user and group mqm on UNIX systems. On Windows systems, they run as the user that started the queue manager. This can be one of:

1. The user who issued the **strmqm** command, or
2. The user under which the IBM MQSeries Service COM server runs

By default, this user is called MUSR_MQADMIN.

If you have not specified a user name and password on the xa_open string, **the user under which the queue manager is running** is used by DB2 to authenticate the xa_open call. If this user (for example, user mqm on UNIX

systems) does not have minimal privileges in the database, the database refuses to authenticate the xa_open call.

The same considerations apply to your application process. If you have not specified a user name and password on the xa_open string, the user under which your application is running is used by DB2 to authenticate the xa_open call that is made during the first MQBEGIN. Again, this user must have minimal privileges in the database for this to work.

For example, give the mqm user connect authority in the mydbname database by issuing the following DB2 commands:

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

See "Security considerations" on page 153 for more information about security.

Change the tp_mon_name parameter

For DB2 for Windows systems only, change the TP_MON_NAME configuration parameter to name the DLL that DB2 uses to call the queue manager for dynamic registration.

Use the command `db2 update dbm cfg using TP_MON_NAME mqmax` to name MQMAX.DLL as the library that DB2 uses to call the queue manager. This must be present in a directory within PATH.

Reset the maxappls parameter

You might need to review your setting for the *maxappls* parameter, which limits the maximum number of applications that can be connected to a database. Refer to "Database connections" on page 144.

Oracle configuration

Do the following:

1. Check environment variable settings.
2. Create the Oracle switch load file.
3. Add resource manager configuration information.
4. Change the Oracle configuration parameters, if necessary.

A current list of levels of Oracle supported by WebSphere MQ is provided at:

`http://www.ibm.com/software/ts/mqseries/platforms/supported.html`

Checking the Oracle environment variable settings

Ensure that your Oracle environment variables are set for queue manager processes ***as well as in*** your application processes. In particular, always set the following environment variables **before** starting the queue manager:

ORACLE_HOME

The Oracle home directory. For example, on UNIX systems, use:

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```

On Windows systems, use:

```
set ORACLE_HOME=c:\oracle\ora81
```

ORACLE_SID

The Oracle SID being used. If you are using Net8 for client/server connectivity, you might not need to set this environment variable. Consult your Oracle documentation.

An example of setting this, on UNIX systems, is:

```
export ORACLE_SID=sid1
```

The equivalent on Windows systems is:

```
set ORACLE_SID=sid1
```

Creating the Oracle switch load file

The easiest way to create the Oracle switch load file is to use the sample file xaswit.mak, which WebSphere MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find xaswit.mak in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Oracle switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak oraswit.dll
```

On AIX, you can find xaswit.mak in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`. On all UNIX systems the directory also contains the C source files that you need to build the switch load file.

Edit xaswit.mak to *uncomment* the lines appropriate to the version of Oracle you are using. Then execute the makefile using the command:

```
make -f xaswit.mak oraswit
```

The generated switch file is placed in your current directory.

Adding resource manager configuration information for Oracle

The next step is to modify the configuration information for the queue manager, as described in "Adding configuration information to the queue manager" on page 145, to declare Oracle as a participant in global units of work. On Windows systems, use the Resources page of the properties for the queue manager (accessed from the WebSphere MQ Services snap-in). On WebSphere MQ for UNIX systems, use the `XAResourceManager` stanza of the qm.ini configuration file.

Figure 12 shows a UNIX sample. We recommend that you add a `LogDir` to the XA open string so that all error and tracing information is logged to the same place.

```
XAResourceManager:
  Name=myoracle
  SwitchFile=/home/myuser/mylibraries/oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp
  ThreadOfControl=PROCESS
```

Figure 12. Sample XAResourceManager entry for Oracle on UNIX platforms

See the *Oracle8 Server Application Developer's Guide* for more information on the xa_open string.

Changing Oracle configuration parameters

For each Oracle database that the queue manager is coordinating, you need to:

Review your maximum sessions

You might need to review your LICENSE_MAX_SESSIONS and PROCESSES settings to take into account the additional connections required by processes belonging to the queue manager. Refer to "Database connections" on page 144 for more details.

Set database privileges

The Oracle user name specified in the xa_open string must have privileges to access the DBA_PENDING_TRANSACTIONS view, as described in the Oracle documentation.

The necessary privilege can be given using the following example command:

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

Sybase configuration

Do the following:

1. Check environment variable settings.
2. Enable Sybase XA support.
3. Create the Sybase switch load file.
4. Add resource manager configuration information.

A current list of levels of Sybase supported by WebSphere MQ is provided at:

`http://www.ibm.com/software/ts/mqseries/platforms/supported.html`

Checking the Sybase environment variable settings

Ensure that your Sybase environment variables are set for queue manager processes ***as well as in*** your application processes. In particular, always set the following environment variables **before** starting the queue manager:

SYBASE

The location of the Sybase product installation. For example, on UNIX systems, use:

```
export SYBASE=/sybase
```

On Windows systems, use:

```
set SYBASE=c:\sybase
```

SYBASE_OCS

The directory under SYBASE where you have installed the Sybase client files. For example, on UNIX systems, use:

```
export SYBASE_OCS=OCS_12-0
```

On Windows systems, use:

```
set SYBASE_OCS=OCS_12-0
```

Enabling Sybase XA support

Within the Sybase XA configuration file (`$SYBASE/$SYBASE_OCS/xa_config`), define a Logical Resource Manager (LRM) for each connection to the Sybase server that is being updated.

An example of the contents of `$SYBASE/$SYBASE_OCS/xa_config` is shown in Figure 13.

```
# The first line must always be a comment

[xa]

  LRM=lrmname
  server=servername
```

Figure 13. Example contents of $SYBASE/$SYBASE_OCS/xa_config

Creating the Sybase switch load file

The easiest way to create the Sybase switch load file is to use the sample files supplied with WebSphere MQ.

On Windows systems, you can find xaswit.mak in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Sybase switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak sybswit.dll
```

On AIX, you can find xaswit.mak in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`. On all UNIX systems the directory also contains the C source files that you need to build the switch load file.

Edit xaswit.mak to *uncomment* the lines appropriate to the version of Sybase you are using. Then execute the makefile using the command:

```
make -f xaswit.mak sybswit
```

The generated switch file is placed in your current directory.

Adding resource manager configuration information for Sybase

The next step is to modify the configuration information for the queue manager, as described in "Adding configuration information to the queue manager" on page 145, to declare Sybase as a participant in global units of work. On Windows systems, use the Resources page of the properties for the queue manager (accessed from the WebSphere MQ Services snap-in). On WebSphere MQ for UNIX systems, use the `XAResourceManager` stanza of the qm.ini configuration file.

Figure 14 shows a UNIX sample, which uses the database associated with the *lrmname* LRM definition in the Sybase XA configuration file, `$SYBASE/$SYBASE_OCS/xa_config`. Include a log file name if you want XA function calls to be logged. :

```
XAResourceManager:
  Name=mysybase
  SwitchFile=/home/myuser/mylibraries/sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
  ThreadOfControl=PROCESS
```

Figure 14. Sample XAResourceManager entry for Sybase on UNIX platforms

Using multi-threaded programs with Sybase

If you are using multi-threaded programs with WebSphere MQ global units of work incorporating updates to Sybase, you ***must*** use the value THREAD for the ThreadOfControl parameter in the relevant XAResourceManager stanza in the qm.ini file or Windows registry entry.

Also ensure that you link your program (and the switch load file) with the threadsafe Sybase libraries (the _r versions), and that you use the symbol `sybase_TXS_xa_switch` for the `xa_switch_t` structure in the switch load file.

Multiple database configurations

If you want to configure the queue manager so that updates to multiple databases can be included within global units of work, add an XAResourceManager stanza for each database.

If the databases are all managed by the same database manager, each stanza defines a separate database. Each stanza specifies the same *SwitchFile*, but the contents of the *XAOpenString* are different because it specifies the name of the database being updated. For example, the stanzas shown in Figure 15 configure the queue manager with the DB2 databases *MQBankDB* and *MQFeeDB* on UNIX systems.

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=DB2 MQFeeDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQFeeDB
```

Figure 15. Sample XAResourceManager entries for multiple DB2 databases

If the databases to be updated are managed by different database managers, add an XAResourceManager stanza for each. In this case, each stanza specifies a different *SwitchFile*. For example, if *MQFeeDB* is managed by Oracle instead of DB2, use the following stanzas on UNIX systems:

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=Oracle MQFeeDB
  SwitchFile=/usr/bin/oraswit
  XAOpenString=Oracle_XA+Acc=P/scott/tiger+SesTm=35
       +LogDir=/tmp/ora.log+DB=MQFeeDB
```

Figure 16. Sample XAResourceManager entries for a DB2 and Oracle database

In principle, there is no limit to the number of database instances that can be configured with a single queue manager.

Security considerations

The following information is provided for guidance only. In all cases, refer to the documentation provided with the database manager to determine the security implications of running your database under the XA model.

An application process denotes the start of a global unit of work using the **MQBEGIN** verb. The first **MQBEGIN** call that an application issues connects to all participating databases by calling their client library code at the xa_open entry point. All the database managers provide a mechanism for supplying a user ID and password in their XAOpenString. This is the only time that authentication information flows.

Note that, on UNIX platforms, fastpath applications must run with an effective user ID of mqm while making MQI calls.

Administration tasks

In normal operations, only a minimal amount of administration is necessary after you have completed the configuration steps. The administration job is made easier because the queue manager tolerates database managers not being available. In particular this means that:

- The queue manager can start at any time without first starting each of the database managers.
- The queue manager does not need to stop and restart if one of the database managers becomes unavailable.

This allows you to start and stop the queue manager independently from the database server.

Whenever contact is lost between the queue manager and a database, they need to resynchronize when both become available again. Resynchronization is the process by which any in-doubt units of work involving that database are completed. In general, this occurs automatically without the need for user intervention. The queue manager asks the database for a list of units of work that are in doubt. It then instructs the database to either commit or roll back each of these in-doubt units of work.

When a queue manager starts, it resynchronizes with each database. When an individual database becomes unavailable, only that database needs to be resynchronized the next time that the queue manager notices it is available again.

The queue manager regains contact with a previously unavailable database automatically as new global units of work are started with MQBEGIN. It does this by calling the xa_open function in the database client library. If this xa_open call fails, MQBEGIN returns with a completion code of MQCC_WARNING and a reason code of MQRC_PARTICIPANT_NOT_AVAILABLE. You can retry the MQBEGIN call later.

Do not continue to attempt a global unit of work that involves updates to a database that has indicated failure during MQBEGIN. There will not be a connection to that database through which updates can be made. Your only options are to end the program, or to retry MQBEGIN periodically in the hope that the database might become available again.

Alternatively, you can use the **rsvmqtrn** command to resolve explicitly all in-doubt units of work.

In-doubt units of work

A database might be left with in-doubt units of work if contact with the queue manager is lost after the database manager has been instructed to prepare. Until the database server receives the outcome from the queue manager (commit or roll back), it needs to retain the database locks associated with the updates.

Because these locks prevent other applications from updating or reading database records, resynchronization needs to take place as soon as possible.

If, for some reason, you cannot wait for the queue manager to resynchronize with the database automatically, you can use facilities provided by the database manager to commit or roll back the database updates manually. In the *X/Open Distributed Transaction Processing: The XA Specification*, this is called making a *heuristic* decision. Use it only as a last resort because of the possibility of

compromising data integrity; you might, for example, mistakenly roll back the database updates when all other participants have committed their updates.

It is far better to restart the queue manager, or use the **rsvmqtrn** command when the database has been restarted, to initiate automatic resynchronization.

Displaying outstanding units of work with the dspmqtrn command

While a database manager is unavailable, you can use the **dspmqtrn** command to check the state of outstanding global units of work involving that database.

The **dspmqtrn** command displays only those units of work in which one or more participants are in doubt, awaiting the decision from the queue manager to commit or roll back the prepared updates.

For each of these global units of work, the state of each participant is displayed in the output from **dspmqtrn**. If the unit of work did not update the resources of a particular resource manager, it is not displayed.

With respect to an in-doubt unit of work, a resource manager is said to have done one of the following things:

Prepared
The resource manager is prepared to commit its updates.

Committed
The resource manager has committed its updates.

Rolled-back
The resource manager has rolled back its updates.

Participated
The resource manager is a participant, but has not prepared, committed, or rolled back its updates.

When the queue manager is restarted, it asks each database having an XAResourceManager stanza for a list of its in-doubt global units of work. If the database has not been restarted, or is otherwise unavailable, the queue manager cannot yet deliver to the database the final outcomes for those units of work. The outcome of the in-doubt units of work is delivered to the database at the first opportunity when the database is again available.

In this case, the database manager is reported as being in *prepared* state until such time as resynchronization has occurred.

Whenever the **dspmqtrn** command displays an in-doubt unit of work, it first lists all the possible resource managers that could be participating. These are allocated a unique identifier, *RMId*, which is used instead of the *Name* of the resource managers when reporting their state with respect to an in-doubt unit of work.

Figure 17 on page 156 shows the result of issuing the following command:

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A50600002013746573 74]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Figure 17. Sample dspmqtrn output

The output in Figure 17 shows that there are three resource managers associated with the queue manager. The first is resource manager `0`, which is the queue manager itself. The other two resource manager instances are the `MQBankDB` and `MQFeeDB` DB2 databases.

The example shows only a single in-doubt unit of work. A message is issued for all three resource managers, which means that updates were made to the queue manager and both DB2 databases within the unit of work.

The updates made to the queue manager, resource manager **0**, have been *committed*. The updates to the DB2 databases are in *prepared* state, which means that DB2 must have become unavailable before it was called to commit the updates to the *MQBankDB* and *MQFeeDB* databases.

The in-doubt unit of work has an external identifier called an XID (*transaction id*). This is a piece of data given to DB2 by the queue manager to identify its portion of the global unit of work.

Resolving outstanding units of work with the rsvmqtrn command

The output shown in Figure 17 shows a single in-doubt unit of work in which the commit decision has yet to be delivered to both DB2 databases.

To complete this unit of work, the queue manager and DB2 need to resynchronize when DB2 next becomes available. The queue manager uses the start of new units of work as an opportunity to regain contact with DB2. Alternatively, you can instruct the queue manager to resynchronize explicitly using the **rsvmqtrn** command.

Do this soon after DB2 has been restarted, so that any database locks associated with the in-doubt unit of work are released as quickly as possible. Use the -a option, which tells the queue manager to resolve all in-doubt units of work. In the following example, DB2 has restarted, so the queue manager can resolve the in-doubt unit of work:

```
> rsvmqtrn -m MY_QMGR -a
Any in-doubt transactions have been resolved.
```

Mixed outcomes and errors

Although the queue manager uses a two-phase commit protocol, this does not completely remove the possibility of some units of work completing with mixed outcomes. This is where some participants commit their updates and some back out their updates.

Units of work that complete with a mixed outcome have serious implications because shared resources that should have been updated as a single unit of work are no longer in a consistent state.

Mixed outcomes are mainly caused when heuristic decisions are made about units of work instead of allowing the queue manager to resolve in-doubt units of work itself. Such decisions are outside the queue manager's control.

Whenever the queue manager detects a mixed outcome, it produces FFST information and documents the failure in its error logs, with one of two messages:

- If a database manager rolls back instead of committing:

```
AMQ7606 A transaction has been committed but one or more resource
        managers have rolled back.
```

- If a database manager commits instead of rolling back:

```
AMQ7607 A transaction has been rolled back but one or more resource
        managers have committed.
```

Further messages identify the databases that are heuristically damaged. It is then your responsibility to locally restore consistency to the affected databases. This is a complicated procedure in which you need first to isolate the update that has been wrongly committed or rolled back, then to undo or redo the database change manually.

Changing configuration information

After the queue manager has successfully started to coordinate global units of work, do not change any of the resource manager configuration information.

If you need to change the configuration information you can do so at any time, but the changes do not take effect until after the queue manager has been restarted.

If you remove the resource manager configuration information for a database, you are effectively removing the ability for the queue manager to contact that database manager.

Never change the *Name* attribute in any of your resource manager configuration information. This attribute uniquely identifies that database manager instance to the queue manager. If you change this unique identifier, the queue manager assumes that the database has been removed and a completely new instance has been added. The queue manager still associates outstanding units of work with the old *Name*, possibly leaving the database in an in-doubt state.

Removing database manager instances: If you need to remove a database from your configuration permanently, ensure that the database is not in doubt before you restart the queue manager. Database products provide commands for listing in-doubt transactions. If there are any in-doubt transactions, first allow the queue manager to resynchronize with the database. Do this by starting the queue manager. You can verify that resynchronization has taken place by using the **rsvmqtrn** command or the database's own command for viewing in-doubt units of work. Once you are satisfied that resynchronization has taken place, end the queue manager and remove the database's configuration information.

If you fail to observe this procedure the queue manager still remembers all in-doubt units of work involving that database. A warning message, AMQ7623, is issued every time the queue manager is restarted. If you are never going to configure this database with the queue manager again, use the -r option of the **rsvmqtrn** command to instruct the queue manager to forget about the database's

participation in its in-doubt transactions. The queue manager forgets about such transactions only when in-doubt transactions have been completed with all participants.

There are times when you might need to remove some resource manager configuration information temporarily. On UNIX systems this is best achieved by commenting out the stanza so that it can be easily reinstated at a later time. You might decide to do this if there are errors every time the queue manager contacts a particular database or database manager. Temporarily removing the resource manager configuration information concerned allows the queue manager to start global units of work involving all the other participants. Here is an example of a commented-out `XAResourceManager` stanza follows:

```
# This database has been temporarily removed
#XAResourceManager:
#  Name=DB2 MQBankDB
#  SwitchFile=/usr/bin/db2swit
#  XAOpenString=MQBankDB
```

Figure 18. Commented- out XAResourceManager stanza on UNIX systems

On Windows systems, use the WebSphere MQ Services snap-in to delete the information about the database manager instance. Take great care to type in the correct name in the *Name* field when reinstating it. If you mistype the name, you may face in-doubt problems, as described in "Changing configuration information" on page 157.

XA dynamic registration

The XA specification provides a way of reducing the number of `xa_*` calls that a transaction manager makes to a resource manager. This optimization is known as *dynamic registration*. Dynamic registration is supported by DB2. Other databases might support it; consult the documentation for your database product for details.

Why is the dynamic registration optimization useful? In your application, some global units of work might contain updates to database tables; others might not contain such updates. When no persistent update has been made to a database's tables, there is no need to include that database in the commit protocol that occurs during MQCMIT.

Whether or not your database supports dynamic registration, your application calls `xa_open` during the first MQBEGIN call on a WebSphere MQ connection. It also calls `xa_close` on the subsequent MQDISC call. The pattern of subsequent XA calls depends on whether the database supports dynamic registration:

If your database does not support dynamic registration...

Every global unit of work involves several XA function calls made by WebSphere MQ code into the database client library, regardless of whether you made a persistent update to the tables of that database within your unit of work. These include:

- `xa_start` and `xa_end` from the application process. These are used to declare the beginning and end of a global unit of work.
- `xa_prepare`, `xa_commit`, and `xa_rollback` from the queue manager agent process, amqzlaa0. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

In addition, the queue manager agent process also calls xa_open during the first MQBEGIN.

If your database supports dynamic registration...

The WebSphere MQ code makes only those XA function calls that are necessary. For a global unit of work that has **not** involved persistent updates to database resources, there are **no** XA calls to the database. For a global unit of work that **has** involved such persistent updates, the calls are to:

- xa_end from the application process to declare the end of the global unit of work.
- xa_prepare, xa_commit, and xa_rollback from the queue manager agent process, amqzlaa0. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

For dynamic registration to work, it is vital that the database has a way of telling WebSphere MQ when it has performed a persistent update that it wants to be included in the current global unit of work. WebSphere MQ provides the ax_reg function for this purpose.

The database's client code that runs in your application process finds the ax_reg function and calls it, to *dynamically register* the fact it has done persistent work within the current global unit of work. In response to this ax_reg call, WebSphere MQ records that the database has participated. If this is the first ax_reg call on this WebSphere MQ connection, the queue manager agent process calls xa_open.

The database client code make this ax_reg call when it is running in your process, for example, during an SQL UPDATE call or whatever call in the database's client API is responsible

Error conditions

There is an opportunity here for a confusing failure in the queue manager. Here is a common example. If you forget to set your database environment variables properly before starting your queue manager, the queue manager's calls to xa_open fail. No global units of work can be used.

To avoid this, ensure that you have set the relevant environment variables before starting the queue manager. Review your database product's documentation, and the advice given in "Checking the DB2 environment variable settings" on page 147, "Checking the Oracle environment variable settings" on page 149, and "Checking the Sybase environment variable settings" on page 151.

With all database products, the queue manager calls xa_open once at queue manager startup, as part of the recovery session (as explained in "Administration tasks" on page 154). This xa_open call fails if you set your database environment variables incorrectly, but it does not cause the queue manager to fail to start. This is because the same xa_open error code is used by the database client library to indicate that the database server is unavailable. We do not treat this as a serious error, as the queue manager must be able to start to continue processing data outside global units of work involving that database.

Subsequent calls to xa_open are made from the queue manager during the first MQBEGIN on a WebSphere MQ connection (if dynamic registration is not being used) or during a call by the database client code to the WebSphere MQ-provided ax_reg function (if dynamic registration is being used).

The **timing** of any error conditions (or, occasionally, FFST reports) depends on whether you are using dynamic registration:

- If you are using dynamic registration, your MQBEGIN call could succeed, but your SQL UPDATE (or similar) database call will fail.
- If you are not using dynamic registration, your MQBEGIN call will fail.

Ensure that your environment variables are set correctly in your application and queue manager processes.

Summarizing XA calls

Table 16 lists the calls that are made to the XA functions in a database client library as a result of the various MQI calls that control global units of work. This is not a complete description of the protocol described in the XA specification; we have provided it as a brief overview.

Note that `xa_start` and `xa_end` calls are always called by WebSphere MQ code in the application process, whereas `xa_prepare`, `xa_commit`, and `xa_rollback` are always called from the queue manager agent process, amqzlaa0.

The `xa_open` and `xa_close` calls shown in this table are all made from the application process. The queue manager agent process calls `xa_open` in the circumstances described in "Error conditions" on page 159.

Table 16. Summary of XA function calls

MQI call	XA calls made with dynamic registration	XA calls made without dynamic registration
First MQBEGIN	`xa_open`	`xa_open` `xa_start`
Subsequent MQBEGIN	No XA calls	`xa_start`

MQCMIT (**without** `ax_reg` being called during the current global unit of work)	No XA calls	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (**with** `ax_reg` being called during the current global unit of work)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		Not applicable. No calls are made to ax_reg in non-dynamic mode.

MQBACK (**without** `ax_reg` being called during the current global unit of work)	No XA calls	`xa_end` `xa_rollback`
MQBACK (**with** `ax_reg` being called during the current global unit of work)	`xa_end` `xa_rollback`	Not applicable. No calls are made to ax_reg in non-dynamic mode.
MQDISC, where MQCMIT or MQBACK was called first. If they were not, MQCMIT processing is first done during MQDISC.	`xa_close`	`xa_close`
Notes: 1. For MQCMIT, `xa_commit` is called if `xa_prepare` is successful. Otherwise, `xa_rollback` is called.		

Scenario 2: Other software provides the coordination

In scenario 2, an external transaction manager coordinates global units of work, starting and committing them under control of the transaction manager's API. The MQBEGIN, MQCMIT, and MQBACK verbs are unavailable.

This section describes this scenario, including:

External syncpoint coordination

A global unit of work can also be coordinated by an external X/Open XA-compliant transaction manager. Here the WebSphere MQ queue manager participates in, but does not coordinate, the unit of work.

The flow of control in a global unit of work coordinated by an external transaction manager is as follows:

1. An application tells the external syncpoint coordinator (for example, TXSeries) that it wants to start a transaction.
2. The syncpoint coordinator tells known resource managers, such as WebSphere MQ, about the current transaction.
3. The application issues calls to resource managers associated with the current transaction. For example, the application could issue **MQGET** calls to WebSphere MQ.
4. The application issues a commit or backout request to the external syncpoint coordinator.
5. The syncpoint coordinator completes the transaction by issuing the appropriate calls to each resource manager, typically using two-phase commit protocols.

The supported levels of external syncpoint coordinators that can provide a two-phase commit process for transactions in which WebSphere MQ participates are defined at:

`http://www.ibm.com/software/ts/mqseries/platforms/supported.html`

See the *WebSphere MQ Application Programming Guide* for information about writing and building transactions to be coordinated by an external syncpoint coordinator.

The rest of this chapter describes how to enable external units of work.

The WebSphere MQ XA switch structure

Each resource manager participating in an externally coordinated unit of work must provide an XA switch structure. This structure defines both the capabilities of the resource manager and the functions that are to be called by the syncpoint coordinator.

WebSphere MQ provides two versions of this structure:
- *MQRMIXASwitch* for static XA resource management
- *MQRMIXASwitchDynamic* for dynamic XA resource management

Consult your transaction manager documentation to determine whether to use the static or dynamic resource management interface. Wherever a transaction manager supports it, we recommend that you use dynamic XA resource management.

External syncpoint coordination

In WebSphere MQ for Windows the structures are located in the following libraries:

mqmxa.dll
(contains only the *MQRMIXASwitch* version)

mqmenc.dll
(for use with Encina for Windows)

mqmc4swi.dll
(for use with IBM TXSeries for Windows)

In WebSphere MQ for UNIX systems, these structures are located in the following libraries:

libmqmxa.a
(nonthreaded)

libmqmxa_r.a
(threaded)

Some external syncpoint coordinators (not CICS) require that each resource manager participating in a unit of work supplies its name in the name field of the XA switch structure. The WebSphere MQ resource manager name is MQSeries_XA_RMI.

The syncpoint coordinator defines how the WebSphere MQ XA switch structure links to it. Information about linking the WebSphere MQ XA switch structure with CICS is provided in "Using CICS". For information about linking the WebSphere MQ XA switch structure with other XA-compliant syncpoint coordinators, consult the documentation supplied with those products.

The following considerations apply to using WebSphere MQ with all XA-compliant syncpoint coordinators:

- The xa_info structure passed on any xa_open call by the syncpoint coordinator includes the name of a WebSphere MQ queue manager. The name takes the same form as the queue-manager name passed to the **MQCONN** call. If the name passed on the xa_open call is blank, the default queue manager is used.
- Only one queue manager at a time can participate in a transaction coordinated by an instance of an external syncpoint coordinator. The syncpoint coordinator is effectively connected to the queue manager, and is subject to the rule that only one connection at a time is supported.
- All applications that include calls to an external syncpoint coordinator can connect only to the queue manager that is participating in the transaction managed by the external coordinator (because they are already effectively connected to that queue manager). However, such applications must issue an **MQCONN** call to obtain a connection handle, and an **MQDISC** call before they exit.
- A queue manager with resource updates coordinated by an external syncpoint coordinator must start before the external syncpoint coordinator. Similarly, the syncpoint coordinator must end before the queue manager.
- If your external syncpoint coordinator terminates abnormally, stop and restart your queue manager ***before*** restarting the syncpoint coordinator to ensure that any messaging operations uncommitted at the time of the failure are properly resolved.

Using CICS

CICS is one of the elements of TXSeries. The versions of TXSeries that are XA-compliant (and use a two-phase commit process) are defined at:

http://www.ibm.com/software/ts/mqseries/platforms/supported.html

The CICS two-phase commit process

This process applies to those versions of WebSphere MQ that support an XA-compliant external syncpoint coordinator, as defined at:

http://www.ibm.com/software/ts/mqseries/platforms/supported.html

Requirements of the two-phase commit process: When you use the CICS two-phase commit process with WebSphere MQ, note the following requirements:

- WebSphere MQ and CICS must reside on the same physical machine.
- WebSphere MQ does not support CICS on a WebSphere MQ client.
- You must start the queue manager, with its name specified in the XAD resource definition stanza, ***before*** you attempt to start CICS. Failure to do this will prevent you from starting CICS if you have added an XAD resource definition stanza for WebSphere MQ to the CICS region.
- Only one WebSphere MQ queue manager can be accessed at a time from a single CICS region.
- A CICS transaction must issue an **MQCONN** request before it can access WebSphere MQ resources. The **MQCONN** call must specify the name of the WebSphere MQ queue manager specified on the XAOpen entry of the XAD resource definition stanza for the CICS region. If this entry is blank, the **MQCONN** request must specify the default queue manager.
- A CICS transaction that accesses WebSphere MQ resources must issue an **MQDISC** call from the transaction before returning to CICS. Failure to do this might mean that the CICS application server is still connected, leaving queues open.
- You must ensure that the CICS user ID (cics) is a member of the mqm group, so that the CICS code has the authority to call WebSphere MQ.

  For transactions running in a CICS environment, the queue manager adapts its methods of authorization and determining context as follows:

  – The queue manager queries the user ID under which CICS runs the transaction. This is the user ID checked by the Object Authority Manager, and is used for context information.
  – In the message context, the application type is MQAT_CICS.
  – The application name in the context is copied from the CICS transaction name.

Enabling the CICS two-phase commit process: To enable CICS to use a two-phase commit process to coordinate transactions that include MQI calls, add a CICS XAD resource definition stanza entry to the CICS region.

Here is an example of adding an XAD stanza entry for WebSphere MQ for Windows, where <Drive> is the drive where WebSphere MQ is installed (for example, D:).

```
cicsadd –cxad –r<cics_region> \
     ResourceDescription="MQM XA Product Description" \
     SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
     XAOpen=<queue_manager_name>
```

Here is an example of adding an XAD stanza entry for WebSphere MQ for UNIX systems:

```
cicsadd –cxad –r<cics_region> \
      ResourceDescription="MQM XA Product Description" \
      SwitchLoadFile="/opt/mqm/lib/amqzsc" \
      XAOpen=<queue_manager_name>
```

For information about using the **cicsadd** command, see the *CICS Administration Reference*, or *CICS Administration Guide* for your platform.

Calls to WebSphere MQ can be included in a CICS transaction, and the WebSphere MQ resources will be committed or rolled back as directed by CICS. This support is not available to client applications.

You ***must*** issue an **MQCONN** from your CICS transaction in order to access WebSphere MQ resources, followed by a corresponding **MQDISC** on exit.

Enabling CICS user exits: Before using a CICS user exit, read the *CICS Administration Guide* for your platform.

A CICS user exit *point* (normally referred to as a *user exit*) is a place in a CICS module at which CICS can transfer control to a program that you have written (a user exit *program*), and at which CICS can resume control when your exit program has finished its work.

One of the user exits supplied with CICS is the Task termination user exit (UE014015), invoked at normal and abnormal task termination (after any syncpoint has been taken).

WebSphere MQ supplies a CICS task termination exit in source and executable form:

Table 17. CICS task termination exits

WebSphere MQ for...	Source	Executable
Windows	amqzscgn.c	mqmc1415.dll
UNIX systems	amqzscgx.c	amqzscg

If you are currently using this exit, add the WebSphere MQ calls from the supplied exits to your current exits. Integrate the WebSphere MQ calls into your existing exits at the appropriate place in the program logic. See the comments in the sample source file for help with this.

If you are not currently using this exit, add a CICS PD program definition stanza entry to the CICS region.

Here is an example of adding a PD stanza entry for Windows:

```
cicsadd –cpd –r<cics_region> \
      PathName="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc1415.dll" \
      UserExitNumber=15
```

Here is an example of adding a PD stanza entry for UNIX systems:

```
cicsadd –cpd –r<cics_region> \
      PathName="/opt/mqm/lib/amqzscg" \
      UserExitNumber=15
```

Using the Microsoft Transaction Server (MTS)

MTS (Microsoft Transaction Server) is designed to help users run business logic applications in a typical middle tier server. MTS divides work up into *activities*, which are typically short independent chunks of business logic, such as *transfer funds from account A to account B*. MTS relies heavily on object orientation and in particular on COM; loosely an MTS activity is represented by a COM (business) object.

Note: MTS is the name of the component that you can install (from the Option Pack) on Windows NT. On Windows 2000 and Windows XP, it is known as COM+ and is an integrated part of the operating system. To use COM+ on Windows 2000 and Windows XP, you need Hotfix Q313582 (also known as COM+ Rollup Package 19.1). Throughout this section, we refer to it as MTS.

MTS provides three services to the business object administrator, removing much of the worry from the business object programmer:

- Transaction management
- Security
- Resource pooling

You usually use MTS with front end code that is a COM client to the objects held within MTS, and back end services such as a database, with WebSphere MQ bridging between the MTS business object and the back end.

The front end code can be a standalone program, or an Active Server Page (ASP) hosted by the Microsoft Internet Information Server (IIS). The front end code can reside on the same computer as MTS and its business objects, with connection through COM. Alternatively, the front end code can reside on a different computer, with connection through DCOM. You can use different clients to access the same MTS business object in different situations.

The back end code can reside on the same computer as MTS and its business objects, or on a different computer with connection through any of the WebSphere MQ supported protocols.

For detailed information on using MTS, including how to configure it, and how to develop your applications and object code, read the *WebSphere MQ MTS Support* part of the Help Center.

Chapter 12. The WebSphere MQ dead-letter queue handler

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

Messages can be put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH.

Messages put on the DLQ by a queue manager or a message channel agent always have an MQDLH; applications putting messages on the DLQ must supply an MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

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

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

This chapter contains the following sections:

Invoking the DLQ handler

Invoke the DLQ handler using the **runmqdlq** command. You can name the DLQ you want to process and the queue manager you want to use in two ways:

- As parameters to **runmqdlq** from the command prompt. For example:

```
runmqdlq ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER <qrule.rul
```

- In the rules table. For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
```

The above examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

If you do not specify the DLQ or the queue manager as shown above, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The **runmqdlq** command takes its input from `stdin`; you associate the rules table with **runmqdlq** by redirecting `stdin` from the rules table.

To run the DLQ handler you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. For the DLQ

handler to put messages on queues with the authority of the user ID in the message context, you must also be authorized to assume the identity of other users.

For more information about the **runmqdlq** command, see "runmqdlq (run dead-letter queue handler)" on page 294.

The sample DLQ handler, amqsdlq

In addition to the DLQ handler invoked using the **runmqdlq** command, WebSphere MQprovides the source of a sample DLQ handler, amqsdlq, whose function is similar to that provided by **runmqdlq**. You can customize amqsdlq to provide a DLQ handler that meets your requirements. For example, you might decide that you want a DLQ handler that can process messages without dead-letter headers. (Both the default DLQ handler and the sample, amqsdlq, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

In WebSphere MQ for Windows, the source of amqsdlq is supplied in the directory:

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\bin
```

In WebSphere MQ for UNIX systems, the source of amqsdlq is supplied in the directory:

`/opt/mqm/samp/dlq` (`/usr/mqm/samp/dlq` on AIX)

and the compiled version is supplied in the directory:

`/opt/mqm/samp/bin` (`/usr/mqm/samp/bin` on AIX)

The DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table. Note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (`|`) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName***|'_')**
The name of the DLQ you want to process:

1. Any INPUTQ value you supply as a parameter to the **runmqdlq** command overrides any INPUTQ value in the rules table.
2. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command, but you **do** specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If no DLQ is specified or you specify INPUTQ(' ') in the rules table, the name of the DLQ belonging to the queue manager whose name is supplied as a parameter to the **runmqdlq** command is used.
4. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command or as a value in the rules table, the DLQ belonging to the queue manager named on the INPUTQM keyword in the rules table is used.

INPUTQM (*QueueManagerName* **| ' ')**
The name of the queue manager that owns the DLQ named on the INPUTQ keyword:

1. Any INPUTQM value you supply as a parameter to the **runmqdlq** command overrides any INPUTQM value in the rules table.
2. If you do not specify an INPUTQM value as a parameter to the **runmqdlq** command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval* **| 60)**
The interval, in seconds, at which the DLQ handler should reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (YES | NO | *nnn***)**
Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES The DLQ handler waits indefinitely.

NO The DLQ handler ends when it detects that the DLQ is either empty or contains no messages that it can process.

nnn The DLQ handler waits for *nnn* seconds for new work to arrive before ending, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, invoke it again using triggering. For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

An alternative to including control data in the rules table is to supply the names of the DLQ and its queue manager as input parameters to the **runmqdlq** command. If you specify a value both in the rules table and as input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

If you include a control-data entry in the rules table, it must be the **first** entry in the table.

Rules (patterns and actions)

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
 ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because **MQPUT** and **MQPUT1** were inhibited.

All keywords that you can use on a rule are described in the rest of this section. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched), and then describes the action keywords (those that determine how the DLQ handler is to process a matching message).

The pattern-matching keywords

The pattern-matching keywords, which you use to specify values against which messages on the DLQ are matched, are described below. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData* **| *)**
The *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName* **| *)**
The name of the application that issued the **MQPUT** or **MQPUT1** call, as specified in the *PutApplName* field of the message descriptor MQMD of the message on the DLQ.

APPLTYPE (*PutApplType* **| *)**
The *PutApplType* value, specified in the message descriptor MQMD, of the message on the DLQ.

DESTQ (*QueueName* **| *)**
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName* **| *)**
The name of the queue manager of the message queue for which the message is destined.

FEEDBACK (*Feedback* **| *)**
When the *MsgType* value is MQFB_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that need confirmation of their arrival on their destination queues.

FORMAT (*Format* **| *)**
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType* **| *)**
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that need replies.

PERSIST (*Persistence* **| *)**
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify messages on the DLQ that are persistent.

REASON (*ReasonCode* **| *)**
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName* **| *)**
The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName* **| *)**
The name of the queue manager of the reply-to queue, as specified in the message descriptor, MQMD, of the message on the DLQ.

USERID (*UserIdentifier* **| *)**
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD.

The action keywords

The action keywords, used to describe how a matching message is to be processed, are described below.

ACTION (DISCARD | IGNORE | RETRY | FWD)
The action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Delete the message from the DLQ.

IGNORE
Leave the message on the DLQ.

RETRY
If the first attempt to put the message on its destination queue fails, try again. The RETRY keyword sets the number of tries made to implement an action. The RETRYINT keyword of the control data controls the interval between attempts.

FWD Forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword.

FWDQ (*QueueName* **| &DESTQ | &REPLYQ)**
The name of the message queue to which to forward the message when ACTION (FWD) is requested.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, specify REPLYQ (?*) in the message pattern.

FWDQM (*QueueManagerName* **| &DESTQM | &REPLYQM | ' ')**
The queue manager of the queue to which to forward a message.

QueueManagerName
The name of the queue manager of the queue to which to forward a message when ACTION (FWD) is requested.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' ' FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (YES | NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF | CTX)
The authority with which messages should be put by the DLQ handler:

DEF Put messages with the authority of the DLQ handler itself.

CTX Put the messages with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (*RetryCount* **| 1)**
The number of times, in the range 1–999 999 999, to try an action (at the interval specified on the RETRYINT keyword of the control data). The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Rules table conventions

The rules table must adhere to the following conventions regarding its syntax, structure, and contents:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included only once in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- There can be any number of blanks at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For reasons of portability, the significant length of a line must not be greater than 72 characters.

- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (–) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

  ```
  APPLNAME('ABC+
    D')
  ```

  results in 'ABCD', and

  ```
  APPLNAME('ABC-
    D')
  ```

  results in 'ABC D'.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in quoted values are not considered significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

    These parameters are invalid because they contain no significant characters:

    ```
    FORMAT('')
    FORMAT( )
    FORMAT()
    FORMAT
    ```

  - Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank; you can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
  - Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible

`MSGTYPE('*')`	Any message type is eligible

However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.

- Numeric parameters must be in the range 0–999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8 character field:

`'ABCDEFGH'`	8 characters
`'A*C*E*G*I'`	5 characters excluding asterisks
`'*A*C*E*G*I*K*M*O*'`	8 characters excluding asterisks

- Enclose strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, use two single quotation marks to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

How the rules table is processed

The DLQ handler searches the rules table for a rule whose pattern matches a message on the DLQ. The search begins with the first rule in the table, and continues sequentially through the table. When the DLQ handler finds a rule with a matching pattern, it takes the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it applies that rule. If the first try fails, the DLQ handler tries again until the number of tries matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Notes:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors are flagged at that time. (Error messages issued by the DLQ handler are described in *WebSphere MQ Messages*.) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler restarts.

4. The DLQ handler does not alter the content of messages, the MQDLH, or the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the rules table is designed to eliminate the generation of repetitive errors during validation.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are seen, even if the DLQ is defined as first-in-first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, try to ensure that the DLQ contains as few messages as possible; if messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself can fill up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which simply leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, make the final rule in the table a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This forwards messages that fall through to the final rule in the table to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table

The following example rules table contains a single control-data entry and several rules:

```
*********************************************************************
*        An example rules table for the runmqdlq command            *
*********************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
```

DLQ handler

```
inputqm(' ')  inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.
* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
  action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message.   PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
```

```
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Chapter 13. Supporting the Microsoft Cluster Server (MSCS)

This information applies to WebSphere MQ for Windows only.

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

MSCS supports *failover* of *virtual servers*, which correspond to applications, Web sites, print queues, or file shares (including their disk spindles, files, IP addresses, and so on).

Failover is the process by which MSCS detects a failure in an application on one computer in the cluster, and shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and re-initiates the application there.

This chapter introduces MSCS clusters and describes setting up MSCS support, then tells you how to configure WebSphere MQ for MSCS clustering, in the following sections:

Finally, in "Hints and tips on using MSCS" on page 188, we give some useful hints on using MSCS with WebSphere MQ.

Introducing MSCS clusters

Before we start to look at MSCS clusters, we need to distinguish between them and WebSphere MQ clusters:

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

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

In the rest of this book, *clusters* means WebSphere MQ clusters. In this chapter, *clusters* **always** means MSCS clusters.

Let us start by looking at a two-machine cluster. A two-machine cluster comprises two computers (for example, A and B) that are jointly connected to a network for client access using a *virtual IP address*. They might also be connected to each other by one or more private networks. A and B share at least one disk for the server applications on each to use. There is also another shared disk, which must be a

redundant array of independent disks (*RAID*) Level 1, for the exclusive use of MSCS; this is known as the *quorum* disk. MSCS monitors both computers to check that the hardware and software are running correctly.

In a simple setup such as this, both computers have all the applications installed on them, but only computer A runs with live applications; computer B is just running and waiting. If computer A encounters any one of a range of problems, MSCS shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and re-initiates the application there. This is known as a *failover*. Applications can be made *cluster-aware* so that they interact fully with MSCS and failover gracefully.

A typical setup for a two-computer cluster is as shown in Figure 19.

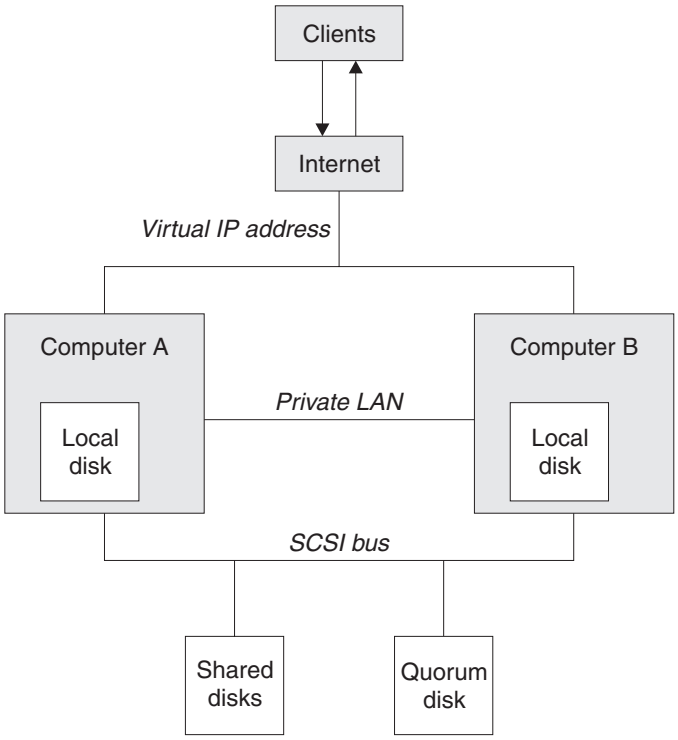


Figure 19. Two-computer MSCS cluster

Each computer can access the shared disk, but only one at a time, under the control of MSCS. In the event of a failover, MSCS switches the access to the other computer. The shared disk itself is usually a RAID, but need not be.

Each computer is connected to the external network for client access, and each has an IP address. However an external client, communicating with this cluster, sees only one *virtual IP address*, and MSCS routes the IP traffic within the cluster appropriately.

MSCS also performs its own communications between the two computers, either over one or more private connections or over the public network, in order to monitor their states using the heartbeat, sunchronize their databases, and so on.

Setting up WebSphere MQ for MSCS clustering

You configure WebSphere MQ for clustering by making the queue manager the unit of failover to MSCS. You define a queue manager as a resource to MSCS, which can then monitor it, and transfer it to another computer in the cluster if there is a problem.

To set your system up for this, you start by installing WebSphere MQ on each computer in the cluster. *WebSphere MQ for Windows, V5.3 Quick Beginnings* tells you how to do this.

The queue managers themselves need to exist only on the computer on which you create them. In the event of a failover, the MSCS initiates the queue managers on the other computer. The queue managers, however, must have their log and data files on a cluster shared disk, and not on a local drive. If you have a queue manager already installed on a local drive, you can migrate it using a tool provided with WebSphere MQ; see "Moving a queue manager to MSCS storage" on page 184. If you want to create new queue managers for use with MSCS, see "Creating a queue manager for use with MSCS" on page 183.

After installation and migration, use the MSCS Cluster Administrator to make MSCS aware of your queue managers; see "Putting a queue manager under MSCS control" on page 185.

If you decide to remove a queue manager from MSCS control, use the procedure described in "Removing a queue manager from MSCS control" on page 187.

Setup symmetry

When an application switches from one node to the other it must behave in the same way, regardless of node. The best way of ensuring this is to make the environments identical. If you can, set up a cluster with identical hardware, operating system software, product software, and configuration on each computer. In particular, ensure that all the required software installed on the two computers is identical in terms of versions, levels, CSDs, SupportPac™s, paths and exits (as described *WebSphere MQ for Windows, V5.3 Quick Beginnings*), and that there is a common namespace (security environment) as described in "MSCS security".

MSCS security

Start by making sure you have identical software installations on each computer in the cluster, as described in *WebSphere MQ for Windows, V5.3 Quick Beginnings*.

For successful MSCS security, follow these guidelines:

- Create a common namespace (security environment) across the cluster.
- Make the nodes of the MSCS cluster members of a domain, within which the user account that is the *cluster owner* is a domain account.
- Make the other user accounts on the cluster also domain accounts, so that they are available on both nodes. This is automatically the case if you already have a domain, and the accounts relevant to WebSphere MQ are domain accounts. If you do not currently have a domain, consider setting up a *mini-domain* to cater for the cluster nodes and relevant accounts. Your aim is to make your cluster of two computers look like a single computing resource.

  Remember that an account that is local to one computer does not exist on the other one. Even if you create an account with the same name on the other

computer, its security identifier (SID) is different, so, when your application is moved to the other node, the permissions do not exist on that node.

During a failover or move, WebSphere MQ MSCS support ensures that all files that contain queue manager objects have equivalent permissions on the destination node. Explicitly, the code checks that the Administrators and mqm groups, and the SYSTEM account, have full control, and that if `Everyone` had read access on the old node, that permission is added on the destination node.

You can use a domain account to run your WebSphere MQ Service. Make sure that it exists in the local mqm group on each computer in the cluster.

Using multiple queue managers with MSCS

If you are running more than one queue manager on a computer, you can choose one of the following setups:

- One shared disk for all queue managers to use. In this configuration, if a problem occurs with any queue manager, all the queue managers failover to the other computer as a group.
- Each queue manager with its own shared disk. In this configuration, if a problem occurs with one queue manager, it alone fails over to the other computer without affecting the other queue managers.
- A mixture of the first two setups.

Cluster modes

There are two modes in which you might run a cluster system with WebSphere MQ:

- Active/Passive
- Active/Active

Note: If you are using MSCS together with the Microsoft Transaction Server (MTS), you cannot use Active/Active mode.

Active/Passive mode

In Active/Passive mode, computer A has the running application on it, and computer B is backup, only being used when MSCS detects a problem.

You can use this mode with only one shared disk, but, if any application causes a failover, **all** the applications must be transferred as a group (because only one computer can access the shared disk at a time).

You can configure MSCS with A as the *preferred* computer. Then, when computer A has been repaired or replaced and is working properly again, MSCS detects this and automatically switches the application back to computer A.

If you run more than one queue manager, consider having a separate shared disk for each. Then put each queue manager in a separate group in MSCS. In this way, any queue manager can failover to the other computer without affecting the other queue managers.

Active/Active mode

In Active/Active mode, computers A and B both have running applications, and the groups on each computer are set to use the other computer as backup. If a failure is detected on computer A, MSCS transfers the state data to computer B, and reinitiates the application there. computer B then runs its own application and A's.

For this setup you need at least two shared disks. You can configure MSCS with A as the preferred computer for A's applications, and B as the preferred computer for B's applications. After failover and repair, each application automatically ends up back on its own computer.

For WebSphere MQ this means that you could, for example, run two queue managers, one on each of A and B, with each exploiting the full power of its own computer. After a failure on computer A, both queue managers would run on computer B. This would mean sharing the power of the one computer, with a reduced ability to process large quantities of data at speed. However, your critical applications would still be available while you find and repair the fault on A.

Creating a queue manager for use with MSCS

This procedure ensures that a new queue manager is created in such a way that it is suitable for preparing and placing under MSCS control.

You start by creating the queue manager with all its resources on a local drive, and then migrate the log files and data files to a shared disk. (You can reverse this operation.) Do **not** attempt to create a queue manager with its resources on a shared drive.

You can create a queue manager for use with MSCS in two ways, either from a command prompt, or in the WebSphere MQ Explorer. The advantage of doing using a command prompt is that the queue manager is created *stopped* and set to *manual startup*, which is ready for MSCS. (The WebSphere MQ Explorer automatically starts a new queue manager and sets it to automatic startup after creation. You have to change this.)

Creating a queue manager from a command prompt

1. Ensure that you have the environment variable `MQS_PREFIX` set to refer to a local drive, for example `C:\WebSphere MQ`. If you change this, reboot the machine so that the System account picks up the change. If you do not set the variable, the queue manager is created in the WebSphere MQ default directory for queue managers.
2. Create the queue manager using the **crtmqm** command. For example, to create a queue manager called `mscs_test` in the default directory, use:

   ```
   crtmqm mscs_test
   ```

3. Proceed to "Moving a queue manager to MSCS storage" on page 184.

Creating a queue manager using the WebSphere MQ Explorer

1. Start the WebSphere MQ Explorer from the Start menu, and expand the folders to find the `queue managers` folder.
2. Right-click the `queue managers` folder, select `New` and complete the dialogs, putting the log path and data path onto a local drive.
3. When the queue manager has been created and started, right-click it in the `queue managers` folder and select `Stop`.
4. Open the WebSphere MQ Services snap-in administration interface and set the startup attribute of the queue manager to manual. (Using the Console menu you can add the WebSphere MQ Services snap-in to the MMC console you already have open for the WebSphere MQ Explorer snap-in.) For more information see "Manual startup" on page 188.
5. Proceed to "Moving a queue manager to MSCS storage" on page 184.

Moving a queue manager to MSCS storage

This procedure configures an existing queue manager to make it suitable for putting under MSCS control. To achieve this, you move the log files and data files to shared disks to make them available to the other computer in the event of a failure. For example, the existing queue manager might have paths such as `C:\WebSphere MQ\log\<QMname>` and `C:\WebSphere MQ\qmgrs\<QMname>`. Do ***not*** try to move the files by hand; use the utility program supplied as part of WebSphere MQ MSCS Support as described below.

The procedure is:

1. Shut down the queue manager, and check that there are no errors.
2. If the queue manager's log files or queue files are already stored on a shared disk, skip the rest of this procedure and proceed directly to "Putting a queue manager under MSCS control" on page 185.
3. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 190 for why this is important).
4. If you already have a suitable shared disk resource proceed to step 6. Otherwise, using the MSCS Cluster Administrator to create a resource of type *shared disk* with sufficient capacity to store the queue manager log files and data (queue) files.
5. Test the shared disk by using the MSCS Cluster Administrator to move it from one cluster node to the other and back again.
6. Make sure that the shared disk is online on the cluster node where the queue manager log and data files are stored locally.
7. Run the utility program to move the queue manager as follows:

   ```
   hamvmqm /m qmname /dd "e:\WebSphere MQ" /ld e:\WebSphere MQ\log"
   ```

   substituting your queue manager name for *qmname*, your shared disk drive letter for *e*, and your chosen directory for *WebSphere MQ*. The directories are created if they do not already exist.
8. Test the queue manager to ensure that it works, using the WebSphere MQ Explorer. For example:
   a. Expand the folder with the name of the queue manager.
   b. Right-click the queue manager folder and select `Start`.
   c. Right-click the queues folder, select `New->Local`, and give the queue a name.
   d. Right-click the queue in the results pane, and use `Put test message` to put several test messages.
   e. Double-click the queue to display the test messages.
   f. Right-click the queue and select `Tasks->Clear messages`.
   g. Right-click the queue and select `Delete`.
   h. Right-click the queue manager folder and select `Stop`.
9. As WebSphere MQ Administrator ensure that the startup attribute of the queue manager is set to manual. Use the WebSphere MQ Services administration interface.
10. Proceed to "Putting a queue manager under MSCS control" on page 185.

Putting a queue manager under MSCS control

Before you put a queue manager under MSCS control:

1. Ensure that WebSphere MQ and its MSCS Support is installed on both machines in the cluster and that the software on each computer is identical, as described in "Setting up WebSphere MQ for MSCS clustering" on page 181.
2. If you have not yet created the queue manager, see "Creating a queue manager for use with MSCS" on page 183.
3. If you have created the queue manager, or it already exists, ensure that you have carried out the procedure in "Moving a queue manager to MSCS storage" on page 184.
4. Stop the queue manager, if it is running, using either a command prompt or the WebSphere MQ Explorer.
5. Test MSCS operation of the shared drives before going on to the procedure below.

To place a queue manager under MSCS control:

1. Log in to the cluster node computer hosting the queue manager, or log in to a remote workstation as a user with cluster administration permissions, and connect to the cluster node hosting the queue manager.
2. Start the MSCS Cluster Administrator.
3. Open a connection to the cluster.
4. Create an MSCS group to be used to contain the resources for the queue manager. Name the group in such a way that it is obvious which queue manager it relates to. For example, you might decide to call the group QM1-Group. Each group must contain only one queue manager, as described in "Using multiple queue managers with MSCS" on page 182.

   Use the group for all the remaining steps.
5. Create a resource instance for each of the SCSI logical drives that the queue manager uses.

   You can use one drive to store both the logs and queue files, or you can split them up across drives. In either case, if each queue manager has its own shared disk, ensure that all drives used by this queue manager are exclusive to this queue manager, that is, that nothing else relies on the drives. Also ensure that you create a resource instance for every drive that the queue manager uses.

   The resource type for a drive depends on the SCSI support you are using; refer to your SCSI adapter instructions. There might already be groups and resources for each of the shared drives. If so, you do not need to create the resource instance for each drive. Just move it from its current group to the one created for the queue manager.

   For each drive resource, set possible owners to both nodes. Set dependent resources to none.
6. Create a resource instance for the IP address.

   Create an IP address resource (resource type *IP Address*). This address should be an unused IP address to be used by clients and other queue managers to connect to the *virtual* queue manager. This IP address is not the normal (static) address of either node; it is an additional address that *floats* between them. Although MSCS handles the routing of this address, it does **not** verify that the address can be reached.
7. Create a resource instance for the queue manager.

Create a resource of type *IBM WebSphere MQ MSCS*; the wizard prompts you for various items, including the following:

- `Name`; choose a name that makes it easy to identify which queue manager it is for.
- `Add to group`; use the group that you created
- `Run in a separate Resource Monitor`; for better isolation
- `Possible owners`; set both nodes
- `Dependencies`; add the drive and IP address for this queue manager
- `Parameters`; as follows:
  - `QueueManagerName` (required); the name of the queue manager that this resource is to control. This queue manager must exist on the local computer.
  - `PostOnlineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from offline to online. For more details see "PostOnlineCommand and PreOfflineCommand" on page 190.
  - `PreOfflineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from online to offline. For more details see "PostOnlineCommand and PreOfflineCommand" on page 190.

8. Optionally, set a preferred node (but note the comments in "Using preferred nodes" on page 191).
9. The *Failover Policy* (as defined in the properties for the group) is set by default to sensible values, but you can tune the thresholds and periods that control *Resource Failover* and *Group Failover* to match the loads placed on the queue manager.
10. Test the queue manager by bringing it online in the MSCS Cluster Administrator and subjecting it to a test workload. If you are experimenting with a test queue manager, use the WebSphere MQ Explorer. For example:
    a. Expand the folder with the name of the queue manager.
    b. Select one of the queues, or, to create a test one, right-click the queues folder, select `New->Local`, and give the queue a name.
    c. Right-click the queue in the results pane, and use `Put test message` to put several test messages.
    d. Double-click the queue to display the test messages.
    e. Right-click the queue and select `Tasks->Clear messages`.
    f. If you created a test queue in (b), right-click the queue and select `Delete`.
11. Test that the queue manager can be taken offline and back online using the MSCS Cluster Administrator.
12. Simulate a failover.

    In the MSCS Cluster Administrator, right-click the group containing the queue manager and select `Move Group`. This can take some minutes to do. (If at other times you just want to move a queue manager to another node quickly, follow the procedure in "Moving a queue manager to MSCS storage" on page 184.) You can also right-click and select `Initiate Failure`; the action (local restart or failover) depends on the current state and the configuration settings.

Removing a queue manager from MSCS control

You can remove queue managers from MSCS control, and return them to manual administration. You do not need to do this for maintenance operations. You can do that by taking a queue manager offline temporarily, using the MSCS Cluster Administrator. Removing a queue manager from MSCS control is a more permanent change; only do it if you decide that you no longer want MSCS to have any further control of the queue manager.

The procedure is:

1. Take the queue manager resource offline using the MSCS Cluster Administrator. To do this, see "Taking a queue manager offline from MSCS".
2. Destroy the resource instance. This does not destroy the queue manager.
3. Optionally, migrate the queue manager files back from shared drives to local drives. To do this, see "Returning a queue manager from MSCS storage".
4. Test the queue manager.

Taking a queue manager offline from MSCS

The procedure is:

1. Start the MSCS Cluster Administrator.
2. Open a connection to the cluster.
3. Select `Groups`, and open the group containing the queue manager to be moved.
4. Select the queue manager resource.
5. Right-click it and select `Offline`.
6. Wait for completion.

Returning a queue manager from MSCS storage

This procedure configures the queue manager to be back on its computer's local drive, that is, it becomes a *normal* WebSphere MQ queue manager. To achieve this, you move the log files and data files from the shared disks. For example, the existing queue manager might have paths such as `E:\WebSphere MQ\log\<QMname>` and `E:\WebSphere MQ\qmgrs\<QMname>`. Do not try to move the files by hand; use the **hamvmqm** utility program supplied as part of WebSphere MQ MSCS Support as described below:

1. Shut down the queue manager, and check that there are no errors.
2. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 190 for why this is important).
3. Decide which local drive to use and ensure that it has sufficient capacity to store the queue manager log files and data (queue) files.
4. Make sure that the shared disk on which the files currently reside is online on the cluster node to which to move the queue manager log and data files.
5. Run the utility program to move the queue manager as follows:

   ```
   hamvmqm /m qmname /dd "c:\WebSphere MQ" /ld "c:\WebSphere MQ\log"
   ```

   substituting your queue manager name for *qmname*, your local disk drive letter for *c*, and your chosen directory for *WebSphere MQ* (the directories are created if they do not already exist).
6. Test the queue manager to ensure that it works (as described in "Moving a queue manager to MSCS storage" on page 184).

Hints and tips on using MSCS

This section contains some general information to help you use WebSphere MQ support for MSCS effectively.

Verifying that MSCS is working

The task descriptions starting with "Creating a queue manager for use with MSCS" on page 183 assume that you have a running MSCS cluster within which you can create, migrate, and destroy resources. If you want to make sure that you have such a cluster:

1. Using the MSCS Cluster Administrator, create a group.
2. Within that group, create an instance of a generic application resource, specifying the system clock (pathname `C:\winnt\system32\clock.exe` and working directory of `C:\`).
3. Make sure that you can bring the resource online, that you can move the group that contains it to the other node, and that you can take the resource offline.

Using the IBM MQSeries Service

Use the IBM MQSeries Service to monitor and control queue managers, running it on both machines in the cluster. The service has its own administration interface. See "MSCS security" on page 181 for information about user account options.

Once you have your queue manager online in MSCS, if you stop the IBM MQSeries Service for any reason, it automatically stops the queue manager. MSCS sees this as a failure condition.

Custom services

WebSphere MQ allows you to define custom services for a queue manager that the IBM MQSeries Service starts and stops when it starts and stops that queue manager. If you place such a queue manager under MSCS control, the custom services that are coordinated to start and stop with that queue manager can automatically be started and stopped during a failover. However, the registry keys that define such custom services are not stored in the same part of the registry as those for the queue manager, so you need to add them, using the `Advanced properties` button on the `Parameters` property page of the MSCS resource for the queue manager. This allows you to specify which of the service keys to checkpoint as part of running the queue manager under MSCS control.

Do ***not*** include keys that might have an adverse effect on other services that are running in the cluster, especially on the other node. This is particularly important when running in Active/Active mode.

WebSphere MQ also allows you to define custom services that are not attached to any queue manager. These cannot be handled by the WebSphere MQ MSCS resource type, because the unit of failover is a queue manager). Create your own resource type in MSCS to handle any such custom services.

Manual startup

For a queue manager managed by MSCS, you ***must*** set the startup attribute to manual. This ensures that the WebSphere MQ MSCS support can restart the IBM MQSeries Service without immediately starting the queue manager.

The WebSphere MQ MSCS support needs to be able to restart the service so that it can perform monitoring and control, but must itself remain in control of which

queue managers are running, and on which machines. See“Moving a queue manager to MSCS storage” on page 184 for more information.

MSCS and queue managers

This section describes some things to consider about your queue managers when using MSCS, as follows:

- “Creating a matching queue manager on the other node”
- “Default queue managers”
- “Deleting a queue manager”
- “Support for existing queue managers”
- “Telling MSCS which queue managers to manage”
- “Queue manager log files” on page 190
- “Multiple queue managers” on page 190

Creating a matching queue manager on the other node

For clustering to work with WebSphere MQ, you need an identical queue manager on node B for each one on node A. However, you do not need to explicitly create the second one. You can create or prepare a queue manager on one node, move it to the other node as described in “Moving a queue manager to MSCS storage” on page 184, and it is fully duplicated on that node.

Default queue managers

Do not use a default queue manager under MSCS control. A queue manager does not have a property that makes it the default; WebSphere MQ keeps its own separate record. If you move a queue manager set to be the default to the other computer on failover, it does not become the default there. Make all your applications refer to specific queue managers by name.

Deleting a queue manager

Once a queue manager has moved node, its details exist in the registry on both computers. When you want to delete it, do so as normal on one computer, and then run the utility described in “WebSphere MQ MSCS support utility programs” on page 191 to clean up the registry on the other computer.

Support for existing queue managers

You can put an existing queue manager under MSCS control, provided that you can put your queue manager log files and queue files on a disk that is on the shared SCSI bus between the two machines (see Figure 19 on page 180). You need to take the queue manager offline briefly while the MSCS Resource is created.

If you want to create a new queue manager, create it independently of MSCS, test it, then put it under MSCS control. See:

- “Creating a queue manager for use with MSCS” on page 183
- “Moving a queue manager to MSCS storage” on page 184
- “Putting a queue manager under MSCS control” on page 185

Telling MSCS which queue managers to manage

You choose which queue managers are placed under MSCS control by using the MSCS Cluster Administrator to create a resource instance for each such queue manager. This process presents you with a list of resources from which to select the queue manager that you want that instance to manage.

Queue manager log files

When you move a queue manager to MSCS storage, you move its log and data files to a shared disk (for an example see "Moving a queue manager to MSCS storage" on page 184).

Always make backups on separate media before you do this. In the log file of a queue manager there are references to fully-qualified paths. When you migrate the queue manager from a local drive to a shared disk, the paths used by the queue manager change and subsequent log entries refer to the new paths. The older (pre-migration) log entries now refer to paths that no longer exist. You cannot replay the log from a point before the migration.

Before you migrate, shut the queue manager cleanly and take a full backup of the queue files and log files. Should the queue manager resources be damaged by media loss in the future, you can restore the files from the backup. Because you shut the queue manager down cleanly, you do not need to replay log records with sequence numbers earlier than the migration. All post-migration log records have valid paths.

Multiple queue managers

WebSphere MQ MSCS support allows you to run multiple queue managers on each machine and to place individual queue managers under MSCS control.

Always use MSCS to manage clusters

Do not try to perform start and stop operations directly on any clustered queue manager using either the WebSphere MQ Explorer or WebSphere MQ Services user interfaces. Instead, use the MSCS Cluster Administrator to request that MSCS brings the queue manager online or takes it offline. This is partly to prevent possible confusion caused by MSCS reporting that the queue manager is offline, when in fact you have started it outside the control of MSCS. More seriously, stopping a queue manager without using MSCS is detected by MSCS as a failure, initiating failover to the other node.

Working in Active/Active mode

Both computers in the MSCS cluster can run queue managers in Active/Active mode. You do not need to have a completely idle machine acting as standby (but you can, if you want, in Active/Passive Mode). If you plan to use both machines to run workload, provide each with sufficient capacity (processor, memory, secondary storage) to run the entire cluster workload at a satisfactory level of performance.

Note: If you are using MSCS together with Microsoft Transaction Server (MTS), you **cannot** use Active/Active mode. This is because, to use WebSphere MQ with MSCS and MTS:

- Application components that use WebSphere MQ's MTS support must run on the same computer as the Distributed Transaction Coordinator (DTC), a part of MTS.
- The queue manager must also run on the same computer.
- The DTC must be configured as an MSCS resource, and can therefore run on only one of the computers in the cluster at any time.

PostOnlineCommand and PreOfflineCommand

Specify these commands in the Parameters to a resource of type `IBM WebSphere MQ MSCS`. You can use them to integrate WebSphere MQ MSCS support with other

systems or procedures. For example, you could specify the name of a program that sends a mail message, activates a pager, or generates some other form of alert to be captured by another monitoring system.

PostOnlineCommand is invoked when the resource changes from offline to online; PreOfflineCommand is invoked for a change from online to offline. Both commands run under the user account used to run the MSCS Cluster Service; and are invoked asynchronously; WebSphere MQ MSCS support does not wait for them to complete before continuing. This eliminates any risk that they might block or delay further cluster operations.

You can also use these commands to issue WebSphere MQ commands, for example to restart Requester channels. However, because they are *edge-triggered*, the commands are not suited to performing long-running functions, and cannot make assumptions about the current state of the queue manager. For example, the commands cannot assume that the queue manager is online; it is quite possible that, immediately after the queue manager was brought online, an administrator issued an offline command.

If you want to run programs that depend on the state of the queue manager, consider creating instances of the `MSCS Generic Application` resource type, placing them in the same MSCS group as the queue manager resource, and making them dependent on the queue manager resource.

Using preferred nodes

It can be useful when using Active/Active mode to configure a *preferred node* for each queue manager. However, in general it is better not to set a preferred node but to rely on a manual failback. Unlike some other relatively stateless resources, a queue manager can take a while to fail over (or back) from one node to the other. To avoid unnecessary outages, test the recovered node before failing a queue manager back to it. This precludes use of the `immediate` failback setting. You can configure failback to occur between certain times of day.

Probably the safest route is to move the queue manager back manually to the desired node, when you are certain that the node is fully recovered. This precludes use of the `preferred node` option.

Performance benchmarking

How long does it take to fail a queue manager over from one machine to the other? This depends heavily on the amount of workload on the queue manager and on the mix of traffic, that is, how much of it is persistent, within syncpoint, how much committed before the failure, and so on. In our test we have seen failover and failback times of about a minute. This was on a very lightly loaded queue manager and actual times will vary considerably depending on load.

WebSphere MQ MSCS support utility programs

WebSphere MQ support for MSCS includes the following utility programs that you can run at a command prompt:

Register/unregister the resource type

`haregtyp.exe`

After you *unregister* the WebSphere MQ MSCS resource type you can no longer create any resources of that type. MSCS does not let you unregister a resource type if you still have instances of that type within the cluster:

1. Using the MSCS Cluster Administrator, stop any queue managers that are running under MSCS control, by taking them offline as described in "Taking a queue manager offline from MSCS" on page 187.
2. Using the MSCS Cluster Administrator, delete the resource instances.
3. At a command prompt, unregister the resource type by entering the following command:

   ```
   haregtyp /u
   ```

If you want to *register* the type (or re-register it at a later time), enter the following command at a command prompt:

```
haregtyp /r
```

After successfully registering the MSCS libraries, you must reboot the system if you have not done so since installing WebSphere MQ.

Move a queue manager to MSCS storage

`hamvmqm.exe`

See "Moving a queue manager to MSCS storage" on page 184.

Delete a queue manager from a node

`hadltmqm.exe`

Consider the case where you have had a queue manager in your cluster, it has been moved from one node to another, and now you want to destroy it. Use the WebSphere MQ Explorer to delete it on the node where it currently is. The registry entries for it still exist on the other computer. To delete these, enter the following command at a prompt on that computer:

```
hadltmqm /m qmname
```

where `qmname` is the name of the queue manager to remove.

Check and save setup details

`amqmsysn.exe`

This utility presents a dialog showing full details of your WebSphere MQ MSCS Support setup, such as might be requested if you call IBM support. There is an option to save the details to a file.

Part 5. Recovery and problem determination

Chapter 14. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. This means that it must provide a method of tracking the messages in the system, and of recovering messages if the system fails for any reason.

WebSphere MQ ensures that messages are not lost by maintaining records (logs) of the activities of the queue managers that handle the receipt, transmission, and delivery of messages. It uses these logs for three types of recovery:
1. *Restart recovery*, when you stop WebSphere MQ in a planned way.
2. *Crash recovery*, when a failure stops WebSphere MQ.
3. *Media recovery*, to restore damaged objects.

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped, except that any in-flight transactions are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; nonpersistent messages are lost during the process.

The rest of this chapter introduces the concepts of recovery and restart in more detail, and tells you how to recover if problems occur. It covers the following topics:

Making sure that messages are not lost (logging)

WebSphere MQ records all significant changes to the data controlled by the queue manager in a log.

This includes creating and deleting objects (except channels), persistent message updates, transaction states, changes to object attributes, and channel activities. The log contains the information you need to recover all updates to message queues by:
- Keeping records of queue manager changes
- Keeping records of queue updates for use by the restart process
- Enabling you to restore data after a hardware or software failure

What logs look like

A WebSphere MQ log consists of two components:
1. One or more files of log data
2. A log control file

There are a number of log files that contain the data being recorded. You can define the number and size (as explained in Chapter 9, "Configuring WebSphere MQ" on page 89), or take the system default of three files.

In WebSphere MQ for Windows, each of the three files defaults to 1 MB. In WebSphere MQ for UNIX systems, each of the three files defaults to 4 MB.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used.

In WebSphere MQ for Windows, if you have not changed the log path, log files are created in the directory:

```
C:\Program Files\IBM\WebSphere MQ\log\<QMgrName>
```

In WebSphere MQ for UNIX systems, if you have not changed the log path, log files are created in the directory:

```
/var/mqm/log/QmName
```

WebSphere MQ starts with these primary log files, but if the log fills, it allocates *secondary* log files. It does this dynamically and removes them when the demand for log space reduces. By default, up to two secondary log files can be allocated. You can change this default allocation, as described in Chapter 9, "Configuring WebSphere MQ" on page 89.

The log control file

The log control file contains the information needed to monitor the use of log files, such as their size and location, the name of the next available file, and so on.

Note: Ensure that the logs created when you start a queue manager are large enough to accommodate the size and volume of messages that your applications will handle. You will probably need to change the default log numbers and sizes to meet your requirements. For more information, see "Calculating the size of the log" on page 200.

Types of logging

In WebSphere MQ, the number of files that are required for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities: circular logging and linear logging.

Circular logging

Use circular logging if all you want is restart recovery, using the log to roll back transactions that were in progress when the system stopped.

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are full. It then goes back to the first file in the ring and starts again. This continues as long as the product is in use, and has the advantage that you never run out of log files.

WebSphere MQ keeps the log entries required to restart the queue manager without loss of data until they are no longer required to ensure queue manager data recovery. The mechanism for releasing log files for reuse is described in "Using checkpointing to ensure complete recovery" on page 198.

Linear logging

Use linear logging if you want both restart recovery and media or forward recovery (recreating lost or damaged data by replaying the contents of the log).

Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so you can always retrieve any record logged from the time that the queue manager was created.

As disk space is finite, you might have to think about some form of archiving. It is an administrative task to manage your disk space for the log, reusing or extending the existing space as necessary.

The number of log files used with linear logging can be very large, depending on your message flow and the age of your queue manager. However, there are a number of files that are said to be *active*. Active files contain the log entries required to restart the queue manager. The number of active log files is usually the same as the number of primary log files as defined in the configuration files. (See "Calculating the size of the log" on page 200 for information about defining the number.)

The key event that controls whether a log file is termed active or not is a *checkpoint*. A WebSphere MQ checkpoint is a group of log records containing the information to enable a successful restart of the queue manager. Any information recorded previously is not required to restart the queue manager and can therefore be termed inactive. (See "Using checkpointing to ensure complete recovery" on page 198 for further information about checkpointing.)

You must decide when you no longer need inactive log files. You can archive them, or you can delete them if they are no longer of interest to your operation. Refer to "Managing logs" on page 202 for further information about the disposition of log files.

If a new checkpoint is recorded in the second, or later, primary log file, the first file becomes inactive and a new primary file is formatted and added to the end of the primary pool, restoring the number of primary files available for logging. In this way the primary log file pool can be seen to be a current set of files in an ever-extending list of log files. Again, it is an administrative task to manage the inactive files according to the requirements of your operation.

Although secondary log files are defined for linear logging, they are not used in normal operation. If a situation arises when, probably due to long-lived transactions, it is not possible to free a file from the active pool because it might still be required for a restart, secondary files are formatted and added to the active log file pool.

If the number of secondary files available is used up, requests for most further operations requiring log activity will be refused with an MQRC_RESOURCE_PROBLEM return code being returned to the application.

Both types of logging can cope with unexpected loss of power, assuming that there is no hardware failure.

Using checkpointing to ensure complete recovery

Persistent updates to message queues happen in two stages. First, the records representing the update are written to the log, then the queue file is updated. The log files can thus become more up-to-date than the queue files. To ensure that restart processing begins from a consistent point, WebSphere MQ uses checkpoints. A checkpoint is a point in time when the record described in the log is the same as the record in the queue. The checkpoint itself consists of the series of log records needed to restart the queue manager; for example, the state of all transactions (units of work) active at the time of the checkpoint.

WebSphere MQ generates checkpoints automatically. They are taken when the queue manager starts, at shutdown, when logging space is running low, and after every 10 000 operations logged.

As the queues handle further messages, the checkpoint record becomes inconsistent with the current state of the queues.

When WebSphere MQ restarts, it finds the latest checkpoint record in the log. This information is held in the checkpoint file that is updated at the end of every checkpoint. The checkpoint record represents the most recent point of consistency between the log and the data. The data from this checkpoint is used to rebuild the queues as they existed at the checkpoint time. When the queues are re-created, the log is then played forward to bring the queues back to the state they were in before system failure or close down.

WebSphere MQ maintains internal pointers to the head and tail of the log. It moves the head pointer to the most recent checkpoint consistent with recovering message data.

Checkpoints are used to make recovery more efficient, and to control the reuse of primary and secondary log files.

In Figure 20 on page 199, all records before the latest checkpoint, Checkpoint 2, are no longer needed by WebSphere MQ. The queues can be recovered from the checkpoint information and any later log entries. For circular logging, any freed files prior to the checkpoint can be reused. For a linear log, the freed log files no longer need to be accessed for normal operation and become inactive. In the example, the queue head pointer is moved to point at the latest checkpoint, Checkpoint 2, which then becomes the new queue head, Head 2. Log File 1 can now be reused.

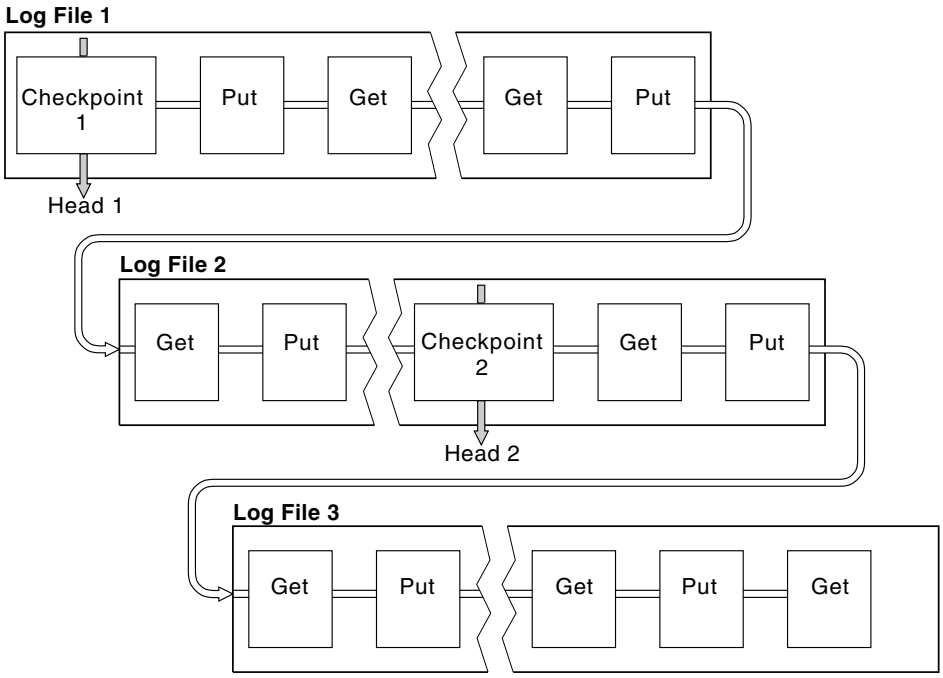


Figure 20. Checkpointing. For simplicity, only the ends of the log files are shown.

Checkpointing with long-running transactions

Figure 21 on page 200 shows how a long-running transaction affects reuse of log files. In the example, a long-running transaction has made an entry to the log, shown as LR 1, after the first checkpoint shown. The transaction does not complete (at point LR 2) until after the third checkpoint. All the log information from LR 1 onwards is retained to allow recovery of that transaction, if necessary, until it has completed.

After the long-running transaction has completed, at LR 2, the head of the log is moved to Checkpoint 3, the latest logged checkpoint. The files containing log records before Checkpoint 3, Head 2, are no longer needed. If you are using circular logging, the space can be reused.

If the primary log files are completely full before the long-running transaction completes, secondary log files are used to avoid the logs getting full.

When the log head is moved and you are using circular logging, the primary log files might become eligible for reuse and the logger, after filling the current file, reuses the first primary file available to it. If you are using linear logging, the log head is still moved down the active pool and the first file becomes inactive. A new primary file is formatted and added to the bottom of the pool in readiness for future logging activities.

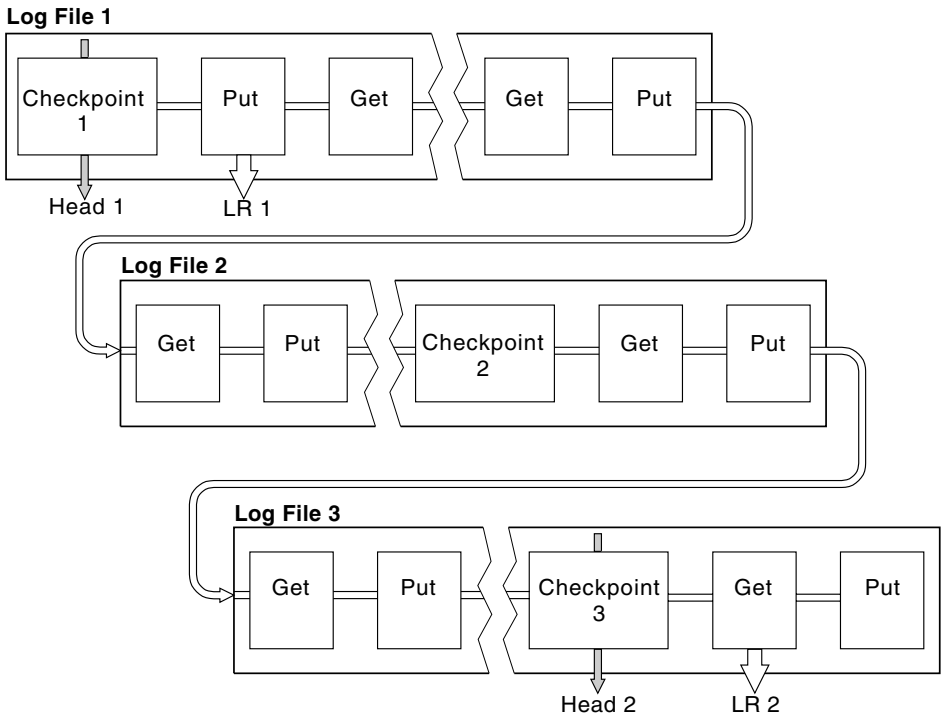


Figure 21. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.

Calculating the size of the log

After deciding whether the queue manager should use circular or linear logging, you need to estimate the size of the log that the queue manager needs. The size of the log is determined by the following log configuration parameters:

LogFilePages
The size of each primary and secondary log file in units of 4K pages

LogPrimaryFiles
The number of preallocated primary log files

LogSecondaryFiles
The number of secondary log files that can be created for use when the primary log files are full

Table 18 shows the amount of data the queue manager logs for various operations. Most queue manager operations need a minimal amount of log space. However, when a persistent message is put to a queue, **all** the message data must be written to the log to make it possible to recover the message. The size of the log depends, typically, on the number and size of the persistent messages the queue manager needs to handle.

Table 18. Log overhead sizes (all values are approximate)

Operation	Size
Put persistent message	750 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, rollback	1000 bytes + 12 bytes for each get or put to be rolled back
Create object	1500 bytes
Delete object	300 bytes
Alter attributes	1024 bytes
Record media image	800 bytes + image The image is divided into segments of 260 000 bytes, each having a 300-byte overhead.
Checkpoint	750 bytes + 200 bytes for each active unit of work Additional data might be logged for any uncommitted puts or gets that have been buffered for performance reasons.

Notes:

1. You can change the number of primary and secondary log files each time the queue manager starts.
2. You cannot change the log file size; you must determine it **before** creating the queue manager.
3. The number of primary log files and the log file size determine the amount of log space that is preallocated when the queue manager is created. Organize this space as a smaller number of larger log files rather than a larger number of small log files.
4. The total number of primary and secondary log files cannot exceed 63, which, in the presence of long-running transactions, limits the maximum amount of log space available to the queue manager for restart recovery. The amount of log space the queue manager might need for media recovery does not share this limit.
5. When *circular* logging is being used, the queue manager reuses primary log space. This means that the queue manager's log can be smaller than the amount of data you have estimated that the queue manager needs to log. The queue manager will, up to a limit, allocate a secondary log file when a log file becomes full, and the next primary log file in the sequence is not available.
6. Primary log files are made available for reuse during a checkpoint. The queue manager takes both the primary and secondary log space into consideration before taking a checkpoint because the amount of log space is running low.

   If you do not define more primary log files than secondary log files, the queue manager might allocate secondary log files before a checkpoint is taken. This makes the primary log files available for reuse.

Managing logs

Over time, some of the log records written become unnecessary for restarting the queue manager. If you are using circular logging, the queue manager reclaims freed space in the log files. This activity is transparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log might fill if a checkpoint has not been taken for a long time, or if a long-running transaction wrote a log record a long time ago. The queue manager tries to take checkpoints often enough to avoid the first problem.

When a long-running transaction fills the log, attempts to write log records fail and some MQI calls return MQRC_RESOURCE_PROBLEM. (Space is reserved to commit or roll back all in-flight transactions, so **MQCMIT** or **MQBACK** should not fail.)

The queue manager rolls back transactions that consume too much log space. An application whose transaction is rolled back in this way cannot perform subsequent **MQPUT** or **MQGET** operations specifying syncpoint under the same transaction. An attempt to put or get a message under syncpoint in this state returns MQRC_BACKED_OUT. The application can then issue **MQCMIT**, which returns MQRC_BACKED_OUT, or **MQBACK** and start a new transaction. When the transaction consuming too much log space has been rolled back, its log space is released and the queue manager continues to operate normally.

If the log fills, message AMQ7463 is issued. In addition, if the log fills because a long-running transaction has prevented the space being released, message AMQ7465 is issued.

Finally, if records are being written to the log faster than the asynchronous housekeeping processes can handle them, message AMQ7466 is issued. If you see this message, increase the number of log files or reduce the amount of data being processed by the queue manager.

What happens when a disk gets full

The queue manager logging component can cope with a full disk, and with full log files. If the disk containing the log fills, the queue manager issues message AMQ6708 and an error record is taken.

The log files are created at their maximum size, rather than being extended as log records are written to them. This means that WebSphere MQ can run out of disk space only when it is creating a new file; it cannot run out of space when it is writing a record to the log. WebSphere MQ always knows how much space is available in the existing log files, and manages the space within the files accordingly.

If you fill the drive containing the log files, you might be able to free some disk space. If you are using a linear log, there might be some inactive log files in the log directory, and you can copy these files to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You might be able to reduce the number of primary or secondary log files so that the log does not outgrow the available space. You

cannot alter the size of the log files for an existing queue manager. The queue manager assumes that all log files are the same size.

Managing log files

If you are using circular logging, ensure that there is sufficient space to hold the log files when you configure your system (see "Log defaults for WebSphere MQ" on page 96 and "Queue manager logs" on page 102). The amount of disk space used by the log does not increase beyond the configured size, including space for secondary files to be created when required.

If you are using a linear log, the log files are added continually as data is logged, and the amount of disk space used increases with time. If the rate of data being logged is high, disk space is consumed rapidly by new log files.

Over time, the older log files for a linear log are no longer needed to restart the queue manager or perform media recovery of any damaged objects. Periodically, the queue manager issues a pair of messages to indicate which of the log files is needed:

- Message AMQ7467 gives the name of the oldest log file needed to restart the queue manager. This log file and all newer log files must be available during queue manager restart.
- Message AMQ7468 gives the name of the oldest log file needed for media recovery.

Any log files older than these do not need to be online. You can copy them to an archive medium such as tape for disaster recovery, and remove them from the active log directory. Any log files needed for media recovery but not for restart can also be off-loaded to an archive.

If any log file that is needed cannot be found, operator message AMQ6767 is issued. Make the log file, and all subsequent log files, available to the queue manager and retry the operation.

Note: When performing media recovery, all the required log files must be available in the log file directory at the same time. Make sure that you take regular media images of any objects you might wish to recover to avoid running out of disk space to hold all the required log files.

Messages AMQ7467 and AMQ7468 can also be issued at the time of running the **rcdmqimg** command. For more information about this command, see "rcdmqimg (record media image)" on page 286.

Log file location

When choosing a location for your log files, remember that operation is severely impacted if WebSphere MQ fails to format a new log because of lack of disk space.

If you are using a circular log, ensure that there is sufficient space on the drive for at least the configured primary log files. Also leave space for at least one secondary log file, which is needed if the log has to grow.

If you are using a linear log, allow considerably more space; the space consumed by the log increases continuously as data is logged.

Ideally, place the log files on a separate disk drive from the queue manager data. This has benefits in terms of performance. It might also be possible to place the log

files on multiple disk drives in a mirrored arrangement. This protects against failure of the drive containing the log. Without mirroring, you could be forced to go back to the last backup of your WebSphere MQ system.

Using the log for recovery

There are several ways that your data can be damaged. WebSphere MQ helps you to recover from:
- A damaged data object
- A power loss in the system
- A communications failure

This section looks at how the logs are used to recover from these problems.

Recovering from power loss or communications failures

WebSphere MQ can recover from both communications failures and loss of power. In addition, it can sometimes recover from other types of problem, such as inadvertent deletion of a file.

In the case of a communications failure, messages remain on queues until they are removed by a receiving application. If the message is being transmitted, it remains on the transmission queue until it can be successfully transmitted. To recover from a communications failure, you can usually restart the channels using the link that failed.

If you lose power, when the queue manager is restarted WebSphere MQ restores the queues to their committed state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when WebSphere MQ stops.

Recovering damaged objects

There are ways in which a WebSphere MQ object can become unusable, for example because of inadvertent damage. You then have to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which objects are damaged.

Media recovery

Media recovery re-creates objects from information recorded in a linear log. For example, if an object file is inadvertently deleted, or becomes unusable for some other reason, media recovery can re-create it. The information in the log required for media recovery of an object is called a *media image*. Media images can be recorded manually, using the **rcdmqimg** command, or automatically in some circumstances.

A media image is a sequence of log records containing an image of an object from which the object itself can be re-created.

The first log record required to re-create an object is known as its *media recovery record*; it is the start of the latest media image for the object. The media recovery record of each object is one of the pieces of information recorded during a checkpoint.

When an object is re-created from its media image, it is also necessary to replay any log records describing updates performed on the object since the last image was taken.

Consider, for example, a local queue that has an image of the queue object taken before a persistent message is put onto the queue. In order to re-create the latest image of the object, it is necessary to replay the log entries recording the putting of the message to the queue, as well as replaying the image itself.

When an object is created, the log records written contain enough information to completely re-create the object. These records make up the object's first media image. Subsequently, at each shutdown, the queue manager records media images automatically as follows:

- Images of all process objects and queues that are not local
- Images of empty local queues

Media images can also be recorded manually using the **rcdmqimg** command, described in "rcdmqimg (record media image)" on page 286. This command writes a media image of the WebSphere MQ object. Once this has been done, only the logs that hold the media image, and all the logs created after this time, are needed to re-create damaged objects. The benefit of doing this depends on such factors as the amount of free storage available, and the speed at which log files are created.

Recovering media images

WebSphere MQ automatically recovers some objects from their media image if it finds that they are corrupt or damaged. In particular, this applies to objects found to be damaged during the normal queue manager startup. If any transaction was incomplete when the queue manager last shut down, any queue affected is also recovered automatically in order to complete the startup operation.

You must recover other objects manually, using the **rcrmqobj** command, which replays the records in the log to re-create the WebSphere MQ object. The object is re-created from its latest image found in the log, together with all applicable log events between the time the image was saved and the time the re-create command was issued. If a WebSphere MQ object becomes damaged, the only valid actions that can be performed are either to delete it or to re-create it by this method. Nonpersistent messages **cannot** be recovered in this way.

See "rcrmqobj (recreate object)" on page 288 for further details of the **rcrmqobj** command.

The log file containing the media recovery record, and all subsequent log files, must be available in the log file directory when attempting media recovery of an object. If a required file cannot be found, operator message AMQ6767 is issued and the media recovery operation fails. If you do not take regular media images of the objects that you want to re-create, you might have insufficient disk space to hold all the log files required to re-create an object.

Recovering damaged objects during start up

If the queue manager discovers a damaged object during startup, the action it takes depends on the type of object and whether the queue manager is configured to support media recovery.

If the queue manager object is damaged, the queue manager cannot start unless it can recover the object. If the queue manager is configured with a linear log, and thus supports media recovery, WebSphere MQ automatically tries to re-create the

queue manager object from its media images. If the log method selected does not support media recovery, you can either restore a backup of the queue manager or delete the queue manager.

If any transactions were active when the queue manager stopped, the local queues containing the persistent, uncommitted messages put or got inside these transactions are also needed to start the queue manager successfully. If any of these local queues is found to be damaged, and the queue manager supports media recovery, it automatically tries to re-create them from their media images. If any of the queues cannot be recovered, WebSphere MQ cannot start.

If any damaged local queues containing uncommitted messages are discovered during startup processing on a queue manager that does not support media recovery, the queues are marked as damaged objects and the uncommitted messages on them are ignored. This is because it is not possible to perform media recovery of damaged objects on such a queue manager and the only action left is to delete them. Message AMQ7472 is issued to report any damage.

Recovering damaged objects at other times

Media recovery of objects is automatic only during startup. At other times, when object damage is detected, operator message AMQ7472 is issued and most operations using the object fail. If the queue manager object is damaged at any time after the queue manager has started, the queue manager performs a preemptive shutdown. When an object has been damaged you can delete it or, if the queue manager is using a linear log, attempt to recover it from its media image using the **rcrmqobj** command (see "rcrmqobj (recreate object)" on page 288 for further details).

Protecting WebSphere MQ log files

Do not remove the log files manually when a WebSphere MQ queue manager is running. If a user inadvertently deletes the log files that a queue manager needs to restart, WebSphere MQ **does not** issue any errors and continues to process data *including persistent messages*. The queue manager shuts down normally, but fails to restart. Media recovery of messages then becomes impossible.

Users with the authority to remove logs that are being used by an active queue manager also have authority to delete other important queue manager resources (such as queue files, the object catalog, and WebSphere MQ executables). They can therefore damage, perhaps through inexperience, a running or dormant queue manager in a way against which WebSphere MQ cannot protect itself.

Exercise caution when conferring super user or mqm authority.

Backing up and restoring WebSphere MQ

Periodically, you might want to take a backup copy of your queue manager data to protect against possible corruption caused by hardware failures. However, because message data is often short-lived, you might choose not to take backups.

Backing up WebSphere MQ

To take a backup copy of a queue manager's data:

1. Ensure that the queue manager is not running. If you try to take a backup of a running queue manager, the backup might not be consistent because of updates in progress when the files were copied.

If possible, stop your queue manager in an orderly way. Try executing **endmqm -w** (a wait shutdown); only if that fails, use **endmqm -i** (an immediate shutdown).

2. Find the directories under which the queue manager places its data and its log files, using the information in the configuration files. For more information about this, see Chapter 9, "Configuring WebSphere MQ" on page 89.

   Note: You might have some difficulty in understanding the names that appear in the directory. The names are transformed to ensure that they are compatible with the platform on which you are using WebSphere MQ. For more information about name transformations, see "Understanding WebSphere MQ file names" on page 18.

3. Take copies of all the queue manager's data and log file directories, including all subdirectories.

   Make sure that you do not miss any files, especially the log control file and the configuration files. Some of the directories might be empty, but you need them all to restore the backup at a later date, so save them too.

4. Preserve the ownerships of the files. For WebSphere MQ for UNIX systems, you can do this with the **tar** command. (If you have queues larger than 2 GB, you cannot use **tar**; for more information, see "Enabling large queues" on page 48.)

Restoring WebSphere MQ

To restore a backup of a queue manager's data:

1. Ensure that the queue manager is not running.
2. Find the directories under which the queue manager places its data and its log files. This information is held in the configuration file.
3. Clear out the directories into which you are going to place the backed-up data.
4. Copy the backed-up queue manager data and log files into the correct places.

Check the resulting directory structure to ensure that you have all the required directories.

See Appendix B, "Directory structure (Windows systems)" on page 479 and Appendix C, "Directory structure (UNIX systems)" on page 481 for more information about WebSphere MQ directories and subdirectories.

Make sure that you have a log control file as well as the log files. Also check that the WebSphere MQ and queue manager configuration files are consistent so that WebSphere MQ can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager will now start.

Note: Even though the queue manager data and log files are held in different directories, back up and restore the directories at the same time. If the queue manager data and log files have different ages, the queue manager is not in a valid state and will probably not start. If it does start, your data is likely to be corrupt.

Recovery scenarios

This section looks at a number of possible problems and indicates how to recover from them.

Disk drive failures

You might have problems with a disk drive containing either the queue manager data, the log, or both. Problems can include data loss or corruption. The three cases differ only in the part of the data that survives, if any.

In ***all*** cases first check the directory structure for any damage and, if necessary, repair such damage. If you lose queue manager data, the queue manager directory structure might have been damaged. If so, re-create the directory tree manually before you restart the queue manager.

Having checked for structural damage, there are a number of things you can do, depending on the type of logging that you use.

- **Where there is major damage to the directory structure or any damage to the log**, remove all the old files back to the `QMgrName` level, including the configuration files, the log, and the queue manager directory, restore the last backup, and restart the queue manager.
- **For linear logging with media recovery**, ensure that the directory structure is intact and restart the queue manager. If the queue manager does not restart, restore a backup. If the queue manager restarts, check, using MQSC commands such as DISPLAY QUEUE, whether any other objects have been damaged. Recover those you find, using the **rcrmqobj** command. For example:

  ```
  rcrmqobj -m QMgrName -t all *
  ```

  where `QMgrName` is the queue manager being recovered. `-t all *` indicates that all objects of any type (except channels) are to be recovered. If only one or two objects have been reported as damaged, you can specify those objects by name and type here.
- **For linear logging with media recovery and with an undamaged log**, you might be able to restore a backup of the queue manager data leaving the existing log files and log control file unchanged. Starting the queue manager applies the changes from the log to bring the queue manager back to its state when the failure occurred.

  This method relies on two things:

  1. You must restore the checkpoint file as part of the queue manager data. This file contains the information determining how much of the data in the log must be applied to give a consistent queue manager.
  2. You must have the oldest log file required to start the queue manager at the time of the backup, and all subsequent log files, available in the log file directory.

  If this is not possible, restore a backup of both the queue manager data and the log, both of which were taken at the same time.
- **For circular logging**, restore the queue manager from the latest backup that you have. Once you have restored the backup, restart the queue manager and check as above for damaged objects. However, because you do not have media recovery, you must find other ways of re-creating the damaged objects.

Damaged queue manager object

If the queue manager object has been reported as damaged during normal operation, the queue manager performs a preemptive shutdown. There are two ways of recovering in these circumstances, depending on the type of logging you use:

- **For linear logging only**, manually delete the file containing the damaged object and restart the queue manager. (You can use the **dspmqfls** command to determine the real, file-system name of the damaged object.) Media recovery of the damaged object is automatic.
- **For circular or linear logging**, restore the last backup of the queue manager data and log and restart the queue manager.

Damaged single object

If a single object is reported as damaged during normal operation:

- **For linear logging**, re-create the object from its media image.
- **For circular logging**, we do not support re-creating a single object.

Automatic media recovery failure

If a local queue required for queue manager startup with a linear log is damaged, and the automatic media recovery fails, restore the last backup of the queue manager data and log and restart the queue manager.

Dumping the contents of the log using the dmpmqlog command

Use the **dmpmqlog** command to dump the contents of the queue manager log. By default all active log records are dumped, that is, the command starts dumping from the head of the log (usually the start of the last completed checkpoint).

The log can usually be dumped only when the queue manager is not running. Because the queue manager takes a checkpoint during shutdown, the active portion of the log usually contains a small number of log records. However, you can use the **dmpmqlog** command to dump more log records using one of the following options to change the start position of the dump:

- Start dumping from the *base* of the log. The base of the log is the first log record in the log file that contains the head of the log. The amount of additional data dumped in this case depends on where the head of the log is positioned in the log file. If it is near the start of the log file, only a small amount of additional data is dumped. If the head is near the end of the log file, significantly more data is dumped.
- Specify the start position of the dump as an individual log record. Each log record is identified by a unique *log sequence number (LSN)*. In the case of circular logging, this starting log record cannot be before the base of the log; this restriction does not apply to linear logs. You might need to reinstate inactive log files before running the command. You must specify a valid LSN, taken from previous **dmpmqlog** output, as the start position.

  For example, with linear logging you can specify the `nextlsn` from your last **dmpmqlog** output. The `nextlsn` appears in `Log File Header` and indicates the LSN of the next log record to be written. Use this as a start position to format all log records written since the last time the log was dumped.
- **For linear logs only**, you can instruct **dmpmqlog** to start formatting log records from any given log file extent. In this case, **dmpmqlog** expects to find this log

file, and each successive one, in the same directory as the active log files. This option does not apply to circular logs, where **dmpmqlog** cannot access log records prior to the base of the log.

The output from the **dmpmqlog** command is the `Log File Header` and a series of formatted log records. The queue manager uses several log records to record changes to its data.

Some of the information that is formatted is only of use internally. The following list includes the most useful log records:

Log File Header
Each log has a single log file header, which is always the first thing formatted by the **dmpmqlog** command. It contains the following fields:

logactive	The number of primary log extents.
loginactive	The number of secondary log extents.
logsize	The number of 4 KB pages per extent.
baselsn	The first LSN in the log extent containing the head of the log.
nextlsn	The LSN of the next log record to be written.
headlsn	The LSN of the log record at the head of the log.
tailsn	The LSN identifying the tail position of the log.
hflag1	Whether the log is CIRCULAR or LOG RETAIN (linear).
HeadExtentID	The log extent containing the head of the log.

Log Record Header
Each log record within the log has a fixed header containing the following information:

LSN	The log sequence number.
LogRecdType	The type of the log record.
XTranid	The transaction identifier associated with this log record (if any). A *TranType* of MQI indicates a WebSphere MQ-only transaction. A *TranType* of XA is involved with other resource managers. Updates involved within the same unit of work have the same *XTranid*.
QueueName	The queue associated with this log record (if any).
Qid	The unique internal identifier for the queue.
PrevLSN	The LSN of the previous log record within the same transaction (if any).

Start Queue Manager
This logs that the queue manager has started.

StartDate	The date that the queue manager started.
StartTime	The time that the queue manager started.

Stop Queue Manager
This logs that the queue manager has stopped.

StopDate	The date that the queue manager stopped.

StopTime The time that the queue manager stopped.

ForceFlag The type of shutdown used.

Start Checkpoint
This denotes the start of a queue manager checkpoint.

End Checkpoint
This denotes the end of a queue manager checkpoint.

ChkPtLSN The LSN of the log record that started this checkpoint.

Put Message
This logs a persistent message put to a queue. If the message was put under syncpoint, the log record header contains a non-null *XTranid*. The remainder of the record contains:

SpcIndex An identifier for the message on the queue. It can be used to match the corresponding **MQGET** that was used to get this message from the queue. In this case a subsequent *Get Message* log record can be found containing the same QueueName and SpcIndex. At this point the SpcIndex identifier can be reused for a subsequent put message to that queue.

Data Contained in the hex dump for this log record is various internal data followed by the Message Descriptor (eyecatcher MD) and the message data itself.

Put Part
Persistent messages that are too large for a single log record are logged as a single *Put Message* record followed by multiple *Put Part* log records.

Data Continues the message data where the previous log record left off.

Get Message
Only gets of persistent messages are logged. If the message was got under syncpoint, the log record header contains a non-null *XTranid*. The remainder of the record contains:

SpcIndex Identifies the message that was retrieved from the queue. The most recent *Put Message* log record containing the same *QueueName* and *SpcIndex* identifies the message that was retrieved.

QPriority The priority of the message retrieved from the queue.

Start Transaction
Indicates the start of a new transaction. A TranType of MQI indicates a WebSphere MQ-only transaction. A TranType of XA indicates one that involves other resource managers. All updates made by this transaction will have the same *XTranid*.

Prepare Transaction
Indicates that the queue manager is prepared to commit the updates associated with the specified *XTranid*. This log record is written as part of a two-phase commit involving other resource managers.

Commit Transaction
Indicates that the queue manager has committed all updates made by a transaction.

Rollback Transaction
This denotes the queue manager's intention to roll back a transaction.

End Transaction
This denotes the end of a rolled-back transaction.

Transaction Table
This record is written during syncpoint. It records the state of each transaction that has made persistent updates. For each transaction the following information is recorded:

XTranid	The transaction identifier.
FirstLSN	The LSN of the first log record associated with the transaction.
LastLSN	The LSN of the last log record associated with the transaction.

Transaction Participants
This log record is written by the XA Transaction Manager component of the queue manager. It records the external resource managers that are participating in transactions. For each participant the following is recorded:

RMName	The name of the resource manager.
RMID	The resource manager identifier. This is also logged in subsequent *Transaction Prepared* log records that record global transactions in which the resource manager is participating.
SwitchFile	The switch load file for this resource manager.
XAOpenString	The XA open string for this resource manager.
XACloseString	The XA close string for this resource manager.

Transaction Prepared
This log record is written by the XA Transaction Manager component of the queue manager. It indicates that the specified global transaction has been successfully prepared. Each of the participating resource managers will be instructed to commit. The *RMID* of each prepared resource manager is recorded in the log record. If the queue manager itself is participating in the transaction a *Participant Entry* with an *RMID* of zero will be present.

Transaction Forget
This log record is written by the XA Transaction Manager component of the queue manager. It follows the *Transaction Prepared* log record when the commit decision has been delivered to each participant.

Purge Queue
This logs the fact that all messages on a queue have been purged, for example, using the MQSC command CLEAR QUEUE.

Queue Attributes
This logs the initialization or change of the attributes of a queue.

Create Object
This logs the creation of a WebSphere MQ object.

ObjName	The name of the object that was created.
UserId	The user ID performing the creation.

Delete Object
This logs the deletion of a WebSphere MQ object.

ObjName The name of the object that was deleted.

See Appendix H, "Example of a log file" on page 501 for an example of the output from a **dmpmqlog** command.

Chapter 15. Problem determination

This chapter suggests reasons for some of the problems you might experience using WebSphere MQ. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

Problem determination is not problem solving. However, the process of problem determination often enables you to solve a problem. For example, if you find that the cause of the problem is an error in an application program, you can solve the problem by correcting the error.

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. This chapter contains these sections:

Preliminary checks

Before you start problem determination in detail, it is worth considering the facts to see if there is an obvious cause of the problem, or a likely area in which to start your investigation. This approach to debugging can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

The cause of your problem could be in:

- WebSphere MQ
- The network
- The application

The sections that follow raise some fundamental questions that you need to consider. As you work through the questions, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause immediately, they could be useful later if you have to carry out a systematic problem determination exercise.

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 one of the following publications to check that you have installed the product correctly, and run the verification procedure:

- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*

- *WebSphere MQ for Windows, V5.3 Quick Beginnings*

Also look at *WebSphere MQ Intercommunication* for information about post-installation configuration of WebSphere MQ.

Are there any error messages?

WebSphere MQ uses error logs to capture messages concerning its own operation, any queue managers that you start, and error data coming from the channels that are in use. Check the error logs to see if any messages have been recorded that are associated with your problem.

See "Error logs" on page 225 for information about the locations and contents of the error logs.

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, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.
- Is it caused by a program? Does it fail on all WebSphere MQ systems and all queue managers, or only on some?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.

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 might have been made to either WebSphere MQ channel definitions or any underlying communications definitions required by your application.
- Do your applications deal with return codes that they might get as a result of any changes you have made?
- Have you changed any component of the operating system that could affect the operation of WebSphere MQ? For example, have you modified the Windows Registry.

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 invokes a rarely-used path in the program.
- Does the application check all return codes?

  Has your WebSphere MQ system been changed, perhaps in a minor way, such that your application does not check the return codes it receives as a result of the change. For example, does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
- Does the application run on other WebSphere MQ systems?

  Could it be that there is something different about the way that this WebSphere MQ system is set up that is causing the problem? For example, have the queues been defined with the same message length or priority?

If the application has not run successfully before

If your application has not yet run successfully, 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 linkage editor, to see if any errors have been reported.

If your application fails to translate, compile, or link-edit into the load library, it will also fail to run if you attempt to invoke it. See the *WebSphere MQ Application Programming Guide* for information about building your application.

If the documentation shows that each of these steps was accomplished without error, consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Common programming errors” for some examples of common errors that cause problems with WebSphere MQ applications.

Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running WebSphere MQ programs. Consider the possibility that the problem with your WebSphere MQ system could be caused by one or more of these errors:

- Assuming that queues can be shared, when they are in fact exclusive.
- Passing incorrect parameters in an MQI call.

- Passing insufficient parameters in an MQI call. This might mean that WebSphere MQ cannot set up completion and reason codes for your application to process.
- Failing to check return codes from MQI requests.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to initialize *MsgId* and *CorrelId* correctly.
- Failing to initialize *Encoding* and *CodedCharSetId* following MQRC_TRUNCATED_MSG_ACCEPTED.

Problems with commands

Be careful when including special characters, for example, back slash (\) and double quote (") characters, in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a \, that is, enter \\ or \" if you want \ or " in your text.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of WebSphere MQ has started.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes, or changed any WebSphere MQ definitions, that might account for the problem?

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it depends on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so these are the times when load-dependent problems are most likely to occur. (If your WebSphere MQ network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Is the problem intermittent?

An intermittent problem could be caused by the way that processes can run independently of each other. For example, a program might issue an MQGET call without specifying a wait option before an earlier process has completed. An intermittent problem might also be seen if your application tries to get a message from a queue 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 you have applied a service update to WebSphere MQ, check that the update action completed successfully and that no error message was produced.

- Did the update have any special instructions?
- Was any test run to verify that the update was applied correctly and completely?
- Does the problem still exist if WebSphere MQ is restored to the previous service level?

- If the installation was successful, check with the IBM Support Center for any PTF error.
- If a PTF has been applied to any other program, consider the effect it might have on the way WebSphere MQ interfaces with it.

Looking at problems in more detail

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 and in the libraries of other licensed programs.

If you have not yet found the cause, start to look at the problem in greater detail. 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.

- "Have you obtained incorrect output?"
- "Have you failed to receive a response from a PCF command?" on page 221
- "Are some of your queues failing?" on page 222
- "Does the problem affect only remote queues?" on page 223
- "Is your application or system running slowly?" on page 223

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.

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, is MAXMSGL sufficiently large?
  - Is the queue enabled for putting?
  - Is the queue already full?
  - Has another application got exclusive access to the queue?
- Are you able to get any messages from the queue?
  - Do you need to take a 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. Ensure that you are waiting long enough for a response.
  - Are you waiting for a specific message that is identified by a message or correlation identifier (*MsgId* or *CorrelId*)?

Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful MQGET call sets both these values to that of the message retrieved, so you might need to reset these values in order to get another message successfully.

Also, check whether you can get other messages from the queue.

- Can other applications get messages from the queue?
- Was the message you are expecting defined as persistent?

  If not, and WebSphere MQ has been restarted, the message has been lost.
- Has another application got exclusive access to the queue?

If you cannot find anything wrong with the queue, and WebSphere MQ is running, check the process that you expected to put the message onto the queue for the following:

- Did the application start?

  If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- Did the application complete correctly?

  Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction issues a successful MQGET call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multiple server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If 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:

- Has your application, or the application that put the message onto the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, the format of the message data might have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear 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 necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

If your application uses an alias queue, check that the alias points to the correct queue.

- Has the trigger information been specified correctly for this queue?

  Check that your application should have started; or should a different application have started?

If these checks do not enable you to solve the problem, check your application logic, both for the program sending the message, and for the program receiving it.

Problems with incorrect output when using distributed queues

If your application uses distributed queues, consider the following points:

- Has 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 is active.

  You can use the MQSC command PING against either the queue manager (PING QMGR) or the channel (PING CHANNEL) to verify that the link is operable.
- Is triggering set on in the sending system?
- Is the message for which you are waiting a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

  If so, check if the message has been put onto the dead-letter queue.

  The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See the *WebSphere MQ Application Programming Reference* for information about the dead-letter queue header structure.
- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap can stop the distributed queuing component. See *WebSphere MQ Intercommunication* for more information about distributed queuing.
- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic conversion occurs when the MQGET call is issued if the format is recognized as one of the built-in formats.

  If the data format is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

  Refer to the *WebSphere MQ Application Programming Guide* for further details of data conversion.

Have you failed to receive a response from a PCF command?

If you have issued a command but have not received a response, consider the following:

- Is the command server running?

  Work with the **dspmqcsv** command to check the status of the command server.

  - If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it.
  - If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *WebSphere MQ Application Programming Reference* for information about the dead-letter queue header structure (MQDLH).

  If the dead-letter queue contains messages, you can use the provided browse sample application (amqsbcg) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.
- Has a message been sent to the error log?

  See "Error logs" on page 225 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?

  If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See the *WebSphere MQ Application Programming Reference* for information about the *WaitInterval* field, and completion and reason codes from MQGET.)
- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to take a syncpoint?

  Unless you have specifically excluded your request message from syncpoint, you need to take a syncpoint before receiving reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the *CorrelId* and *MsgId* fields correctly?

  Set the values of *MsgId* and *CorrelId* in your application to ensure that you receive all messages from the queue.

Try stopping the command server and then restarting it, responding to any error messages that are produced.

If the system still does not respond, the problem could be with either a queue manager or the whole of the WebSphere MQ system. First, try stopping individual queue managers to isolate a failing queue manager. If this does not reveal the problem, try stopping and restarting WebSphere MQ, responding to any messages that are produced in the error log.

If the problem still occurs after restart, contact your IBM Support Center for help.

Are some of your queues failing?

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems:

1. Display the information about each queue. You can use the MQSC command DISPLAY QUEUE to display the information.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, the queue is not being processed. Check that all applications are running normally.

- If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
  - If triggering is being used:
    - Is the trigger monitor running?
    - Is the trigger depth too great? That is, does it generate a trigger event often enough?
    - Is the process name correct?
    - Is the process available and operational?
  - Can the queue be shared? If not, another application could already have it open for input.
  - Is the queue enabled appropriately for GET and PUT?
- If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the MQOPEN call has failed for some reason.

  Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. The values might have changed; the queue might have been open but is now closed.

  You need to check the status at the time you expect to put or get a message.

If you are unable to solve the problem, contact your IBM Support Center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues:

- Check that required channels have started, can be triggered, and any required initiators are running.
- Check that the programs that should be putting messages to the remote queues have not reported problems.
- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually. See *WebSphere MQ Intercommunication* for information about starting channels.

Is your application or system running slowly?

If your application is running slowly, it might be in a loop or waiting for a resource that is not available.

This might also indicate a performance problem. Perhaps your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

A performance problem might be caused by a limitation of your hardware.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly-designed application program is probably to blame. This could appear to be a problem that only occurs when certain queues are accessed.

What next

The following symptoms might indicate that WebSphere MQ is running slowly:

- Your system is slow to respond to MQSC 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 might lie with WebSphere MQ itself. If you suspect this, contact your IBM Support Center for help.

Tuning performance for nonpersistent messages on AIX

If you are using AIX, consider setting your tuning parameter to exploit full performance for nonpersistent messages. To set this tuning parameter, a root user must issue the command `/usr/samples/kernal/vmtune -c 0`.

The effect of this command persists until the next reboot. An AIX administrator can add a line such as `vmtune:2:once:/usr/samples/kernal/vmtune -c 0` to the system `/etc/inittab` file to cause the command to be issued on every reboot. This command is made available by installing the `bos.adt.samples` fileset from the AIX installation CDs.

Normally, nonpersistent messages are kept only in memory, but there are circumstances where AIX can schedule nonpersistent messages to be written to disk. Messages scheduled to be written to disk are unavailable for MQGET until the disk write completes. The suggested tuning command varies this threshold; instead of scheduling messages to be written to disk when 16 kilobytes of data are queued, the write-to-disk occurs only when real storage on the machine becomes close to full.

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 itself, but affect 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

The amount of data in a message can affect the performance of the application that processes the message. To achieve the best performance from your application, send only the essential data in a message. For example, in a request to debit a bank account, the only information that might need to be passed from the client to the server application is the account number and the amount of the debit.

Effect of message persistence

Persistent messages are usually logged. Logging messages reduces the performance of your application, so use persistent messages for essential data only. If the data in a message can be discarded if the queue manager stops or fails, use a nonpersistent message.

Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (*MsgId* and *CorrelId*) 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 can 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, grow and shrink the buffers dynamically to suit the typical message size. Add code so that, when the application issues an MQGET that fails because the buffer is too small for the message size, you resize the buffer and re-issue the MQGET.

Note also that, if you do not set the *MaxMsgLength* attribute explicitly, it defaults to 4 MB, which might be very inefficient.

Frequency of syncpoints

Programs that issue very large numbers of MQPUT or MQGET calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently inaccessible, while other tasks might be waiting to get these messages. This has implications in terms of storage, and in terms of threads tied up with tasks that are attempting to get messages.

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.

Number of threads in use

For WebSphere MQ for Windows, an application might require a large number of threads. Each queue manager process is allocated a maximum allowable number of threads.

Applications might use too many threads. Consider whether the application takes into account this possibility and that it takes actions either to stop or to report this type of occurrence.

Error logs

WebSphere MQ uses a number of error logs to capture messages concerning its own operation of WebSphere MQ, any queue managers that you start, and error data coming from the channels that are in use.

Error logs

The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

In WebSphere MQ for Windows, assuming that WebSphere MQ has been installed in the default location:

- If the queue manager name is known, error logs are located in:
  `c:\Program Files\IBM\WebSphere MQ\qmgrs\qmname\errors`
- If the queue manager name is not known, error logs are located in:
  `c:\Program Files\IBM\WebSphere MQ\qmgrs\@SYSTEM\errors`
- If an error has occurred with a client application, error logs are located on the client's root drive in:
  `c:\Program Files\IBM\WebSphere MQ Client\errors`

In WebSphere MQ for Windows, an indication of the error is also added to the Application Log, which can be examined with the Event Viewer application provided with Windows systems.

In WebSphere MQ for UNIX systems:

- If the queue manager name is known and the queue manager is available, error logs are located in:
  `/var/mqm/qmgrs/qmname/errors`
- If the queue manager is not available, error logs are located in:
  `/var/mqm/qmgrs/@SYSTEM/errors`
- If an error has occurred with a client application, error logs are located on the client's root drive in:
  `/var/mqm/errors`

Log files

At installation time an @SYSTEM errors subdirectory is created in the qmgrs file path. The errors subdirectory can contain up to three error log files named:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

After you have created a queue manager, it creates three error log files when it needs them. These files have the same names as the @SYSTEM ones, that is AMQERR01, AMQERR02, and AMQERR03, and each has a capacity of 256 KB. The files are placed in the errors subdirectory of each queue manager that you create.

As error messages are generated, they are placed in AMQERR01. When AMQERR01 gets bigger than 256 KB it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager's errors files, unless the queue manager is unavailable, or its name is unknown, when channel-related messages are placed in the @SYSTEM errors subdirectory.

To examine the contents of any error log file, use your usual system editor.

Early errors

There are a number of special cases where these error logs have not yet been established and an error occurs. WebSphere MQ attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, because of a corrupt configuration file for example, no location information can be determined, errors are logged to an errors directory that is created at installation time on the root directory (`/var/mqm` or `C:\Program Files\IBM\WebSphere MQ`).

If WebSphere MQ can read its configuration information, and can access the value for the Default Prefix, errors are logged in the errors subdirectory of the directory identified by the Default Prefix attribute. For example, if the default prefix is `C:\Program Files\IBM\WebSphere MQ`, errors are logged in `C:\Program Files\IBM\WebSphere MQ\errors`.

For further information about configuration files, see Chapter 9, "Configuring WebSphere MQ" on page 89.

Note: Errors in the Windows Registry are notified by messages when a queue manager is started.

Ignoring error codes under Windows systems

If you want WebSphere MQ for Windows to ignore error codes, edit the Windows Registry.

The Registry key is:

`HKEY_LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\IgnoredErrorCodes`

The value that you set it to is an array of strings delimited by the NULL character, with each string value relating to the error code that you want ignored. The complete list is terminated with a NULL character.

For example, if you want WebSphere MQ to ignore error codes AMQ3045, AMQ6055, and AMQ8079, set the value to: AMQ3045\0AMQ6055\0AMQ8079\0\0

Any changes you make to a configuration file do not take effect until the next time the queue manager is started.

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national-language enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any. In addition, some operator messages are written to the AMQERR01.LOG file in the queue manager directory, and others to the @SYSTEM directory copy of the error log.

Dead-letter queues

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 issuing the MQSC command DISPLAY QUEUE. If the queue contains messages, use the provided browse sample application (amqsbcg) to browse messages on the queue

using the **MQGET** call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for each message. See "Browsing queues" on page 46 for more information about running this sample and about the kind of output it produces.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue.

Problems might occur if you do not associate a dead-letter queue with each queue manager. For more information about dead-letter queues, see Chapter 12, "The WebSphere MQ dead-letter queue handler" on page 167.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found, and result in *queue manager unavailable* errors. Ensure that the configuration files exist, and that the WebSphere MQ configuration file references the correct queue manager and log directories.

Errors in the Windows Registry are notified by messages when a queue manager is started.

Tracing

This section describes how to produce a trace for WebSphere MQ.

Tracing WebSphere MQ for Windows

In WebSphere MQ for Windows, you enable or modify tracing using the **strmqtrc** control command, described in "strmqtrc (Start trace)" on page 316. To stop tracing, you use the **endmqtrc** control command, described in "endmqtrc (end trace)" on page 285.

For WebSphere MQ for Windows, you can also start and stop trace using the trace icon in the WebSphere MQ Services snap-in.

Selective component tracing on WebSphere MQ for Windows

Use the -t and -x options to control the amount of trace detail to record. By default, **all** trace points are enabled. The -x option enables you to specify the points that you do **not** want to trace. So if, for example, you want to trace only data flowing over communications networks, use:

```
strmqtrc -x all -t comms
```

For a full description of the trace command, see "strmqtrc (Start trace)" on page 316.

Trace files

During the installation process, you can choose the drive on which trace files are to be located. The trace files are always placed in the directory `\<mqmwork>\errors`, where `<mqmwork>` is the directory selected when WebSphere MQ was installed to hold WebSphere MQ data files.

Trace-file names have the following format:

```
AMQpppppp.TRC
```

where *ppppp* is the process identifier (PID) of the process producing the trace.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

An example of WebSphere MQ for Windows trace data

Figure 22 shows an extract from a WebSphere MQ for Windows trace:

```
Process : C:\Program Files\IBM\WebSphere MQ\bin\amqxssvn.exe
Version : 530 Level : p000-L020213
Date : 02/25/02  Time : 16:35:47

Counter  TimeStamp       Process.Thread    Data
=========================================================

0000062F 16:35:47.348386 6278.1       --{  InitProcessInitialisation
00000630 16:35:47.348455 6278.1       ---{  xcsCreateNTSecurityAtts
00000631 16:35:47.348516 6278.1       ----{  xcsRequestThreadMutexSem
00000632 16:35:47.348583 6278.1       ----}  xcsRequestThreadMutexSem (rc=OK)
00000633 16:35:47.348639 6278.1       ----{  xcsInitGlobalSecurityData
00000634 16:35:47.349111 6278.1       ----}  xcsInitGlobalSecurityData (rc=OK)
00000635 16:35:47.349239 6278.1       ----{  xcsReleaseThreadMutexSem
00000636 16:35:47.349261 6278.1       ----}  xcsReleaseThreadMutexSem (rc=OK)
00000637 16:35:47.349275 6278.1       ---}  xcsCreateNTSecurityAtts (rc=OK)
00000638 16:35:47.349303 6278.1       ---{  xcsReleaseThreadMutexSem
00000639 16:35:47.349319 6278.1       ---}  xcsReleaseThreadMutexSem (rc=OK)
0000063A 16:35:47.349344 6278.1       --}  InitProcessInitialisation (rc=OK)
0000063B 16:35:47.349359 6278.1       --{  xcsCreateThreadMutexSem
0000063C 16:35:47.349395 6278.1       --}  xcsCreateThreadMutexSem (rc=OK)
0000063D 16:35:47.349872 6278.1       --{  xcsProgramInit
0000063E 16:35:47.349900 6278.1       --}  xcsProgramInit (rc=OK)
0000063F 16:35:47.350027 6278.1       --{  xcsInitialize
00000640 16:35:47.350048 6278.1       ---{  xcsRequestThreadMutexSem
00000641 16:35:47.350065 6278.1       ---}  xcsRequestThreadMutexSem (rc=OK)
00000642 16:35:47.350079 6278.1       ---{  xihCheckThreadList
00000643 16:35:47.350101 6278.1       ---}  xihCheckThreadList (rc=OK)
00000644 16:35:47.350115 6278.1       ---{  InitPrivateServices
00000645 16:35:47.350165 6278.1       attributes 32768
00000646 16:35:47.350204 6278.1       ----{  xcsCreateThreadMutexSem
00000647 16:35:47.350233 6278.1       ----}  xcsCreateThreadMutexSem (rc=OK)
00000648 16:35:47.350255 6278.1       pid MQ(6) system(6278)
00000649 16:35:47.350337 6278.1       ---}  InitPrivateServices (rc=OK)
0000064A 16:35:47.350360 6278.1       --{  xxxInitialize
0000064B 16:35:47.350977 6278.1       ---{  xcsGetMem
```

Figure 22. Sample WebSphere MQ for Windows trace

Tracing WebSphere MQ for AIX

WebSphere MQ for AIX uses the standard AIX system trace. Tracing is a two-step process:

1. Gathering the data
2. Formatting the results

WebSphere MQ uses two trace hook identifiers:

X'30D' This event is recorded by WebSphere MQ on entry to or exit from a subroutine.

X'30E' This event is recorded by WebSphere MQ to trace data such as that being sent or received across a communications network.

Trace provides detailed execution tracing to help you to analyze problems. IBM service support personnel might ask for a problem to be re-created with trace enabled. The files produced by trace can be **very** large so it is important to qualify a trace, where possible. For example, you can optionally qualify a trace by time and by component.

There are two ways to run trace:

1. Interactively.

   The following sequence of commands runs an interactive trace on the program `myprog` and ends the trace.

   ```
   trace -j30D,30E -o trace.file
   ->!myprog
   ->q
   ```

2. Asynchronously.

   The following sequence of commands runs an asynchronous trace on the program `myprog` and ends the trace.

   ```
   trace -a -j30D,30E -o trace.file
   myprog
   trcstop
   ```

You can format the trace file with the command:

```
trcrpt -t /usr/mqm/lib/amqtrc.fmt trace.file > report.file
```

`report.file` is the name of the file where you want to put the formatted trace output.

Note: **All** WebSphere MQ activity on the machine is traced while the trace is active.

Selective component tracing on WebSphere MQ for AIX

Use the environment variable MQS_TRACE_OPTIONS to activate the high detail and parameter tracing functions individually. Because it enables tracing to be active without these functions, you can use it to reduce the overhead on execution speed when you are trying to reproduce a problem with tracing switched on. Table 19 defines the trace behavior under the various settings of MQS_TRACE_OPTIONS.

Table 19. MQS_TRACE_OPTIONS settings

MQS_TRACE_OPTIONS Value	What will be traced
Unset (default)	Default trace (all except high detail)
0	No WebSphere MQ trace
262148	Entry, exit and parameter trace
786436	Entry, exit, parameter, and high detail trace
4980740	Entry, exit, parameter, high detail, and SSL tracing
3407871	Default trace without parameter trace
3670015	Default trace, including parameter trace
7864319	Default trace, including parameter trace and SSL tracing
4194303	All tracing, including high detail trace

Notes:

1. Set the environment variable MQS_TRACE_OPTIONS only if you have been instructed to do so by your service personnel.
2. Typically MQS_TRACE_OPTIONS must be set in the process that starts the queue manager, and before the queue manager is started, or it is not recognized.
3. Set MQS_TRACE_OPTIONS before tracing starts. If it is set after tracing starts it is not recognized.

SSL trace: If you request SSL trace, note the following:

- SSL trace is written to the directory `/var/mqm/trace`.
- The SSL trace files are AMQ.SSL.TRC and AMQ.SSL.TRC.1.
- You cannot format SSL trace files; send them unchanged to IBM support.

An example of WebSphere MQ for AIX trace data

The following example is an extract of an AIX trace:

```
ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

30D    0.000000000       0.000000   MQS FNC Entry. 71540.1 zcpSendOnPipe
30E    0.000000038       0.000038   Msg Unencumbered (T/F)(0)
30D    0.000000176       0.000138   MQS FNC Exit.............. 51604.55
                                    aqhCheckMsgUnencumbered rc=00000000
30E    0.000000418       0.000242   aqhCheckMsgChains : internal retcode
                                       208007d3

30D    0.000000516       0.000098   MQS FNC Entry.. 71540.14 xcsWaitEventSem
30E    0.000000590       0.000074   MessageSent (24 bytes)
30E    0.000000847       0.000257   aqhCheckMsgChains : internal retcode
                                       208007d3

30E    0.000000936       0.000089   hev=1::0::0-307724 TimeOut(-1)
30E    0.000001173       0.000237   aqhCheckMsgChains : internal retcode
                                       208007d3

30D    0.000001313       0.000140   MQS FNC Entry............. 51604.55
                                    aqtIdxToSpcFn
30D    0.000001395       0.000082   MQS FNC Exit.............. 51604.55
                                    aqtIdxToSpcFn rc=00000000
30D    0.000001439       0.000044   MQS FNC Entry........ 36124.51
                                    xcsCheckProcess
30D    0.000001501       0.000062   MQS FNC Entry............. 51604.55
                                    aqhCheckMsgUnencumbered
30E    0.000001645       0.000144   MQS Data from zcpSendOnPipe Length=0018
                                        5A525354 000007E5 00000000 00000000
                                        ZRST 00000000 00000000
30E    0.000001765       0.000120   pBCrsr (0)
30D    0.000001907       0.000142   MQS FNC Entry.............. 51604.55
                                    aqhInTrans
30D    0.000001997       0.000090   MQS FNC Exit............... 51604.55
                                    aqhInTrans rc=00000000
30D    0.000002025       0.000028   MQS FNC Entry.. 71540.1 xcsResetEventSem
30E    0.000002243       0.000218   Msg Unencumbered (T/F)(0)
30D    0.000002363       0.000120   MQS FNC Exit.............. 51604.55
                                    aqhCheckMsgUnencumbered rc=00000000
30E    0.000002392       0.000029   hev=1::0::0-305876
30D    0.000002522       0.000130   MQS FNC Entry... 71540.14 xlsLockEvent
30E    0.000002630       0.000108   aqhCheckMsgChains : internal retcode
                                       208007d3
```

Figure 23. Sample WebSphere MQ for AIX trace

Tracing WebSphere MQ for HP-UX, WebSphere MQ for Solaris, and WebSphere MQ for Linux for Intel and Linux for zSeries

In WebSphere MQ for HP-UX, Solaris, and Linux, you enable or modify tracing using the **strmqtrc** control command, which is described in "strmqtrc (Start trace)" on page 316. To stop tracing, you use the **endmqtrc** control command, which is described in "endmqtrc (end trace)" on page 285. You can display formatted trace output using the **dspmqtrc** control command, which is described in "dspmqtrc (display formatted trace output)" on page 279.

Selective component tracing on WebSphere MQ for HP-UX, WebSphere MQ for Solaris, and WebSphere MQ for Linux for Intel and Linux for zSeries

Use the -t and -x options to control the amount of trace detail to record. By default, **all** trace points are enabled. The -x option enables you to specify the points you do

not want to trace. So if, for example, you want to trace, for queue manager QM1, only output data associated with using Secure Sockets Layer (SSL) channel security, use:

```
strmqtrc -m QM1 -t ssl
```

For a full description of the trace command, see "strmqtrc (Start trace)" on page 316.

Example trace data for WebSphere MQ for HP-UX, WebSphere MQ for Solaris, and WebSphere MQ for Linux for Intel and Linux for zSeries

Figure 24 shows an extract from a WebSphere MQ for HP-UX trace:

```
ID     ELAPSED_MICROSEC DELTA_MICROSEC  APPL    SYSCALL KERNEL  INTERRUPT

30d     0                0          MQS FNC Exit.......... 24864.1
                                          xllSpinLockRequest rc=00000000
30d     117              117        MQS FNC Entry......... 24864.1
                                          xllSpinLockRelease
30d     150              33         MQS FNC Exit.......... 24864.1
                                          xllSpinLockRelease rc=00000000
30d     180              30         MQS FNC Entry......... 24864.1
                                          xllSpinLockRequest
30d     212              32         MQS FNC Exit.......... 24864.1
                                          xllSpinLockRequest rc=00000000
30d     246              34         MQS FNC Entry......... 24864.1
                                          xllSpinLockRelease
30d     275              29         MQS FNC Exit.......... 24864.1
                                          xllSpinLockRelease rc=00000000
30d     304              29         MQS FNC Exit!........ 24864.1
                                          xllWaitSocketEvent rc=10806020
30d     335              31         MQS FNC Exit!....... 24864.1
                                          xcsWaitEventSem rc=10806020
30d     374              39         MQS FNC Exit!...... 24864.1
                                          zcpReceiveOnPipe rc=20805311
30d     408              34         MQS FNC Entry...... 24864.1 ziiHealthCheck
30d     439              31         MQS FNC Entry....... 24864.1 xcsCheckProcess
30e     506              67         pid(24860)
```

Figure 24. Sample WebSphere MQ for HP-UX trace

Figure 25 on page 234 shows an extract from a WebSphere MQ for Solaris trace:

```
Timestamp        Process.Thread  Trace Data
========================================
 12:25:03.559508     8887.1          Version : 530 Level : p000-L020308
 12:25:03.560647     8887.1          Date : 11/03/02  Time : 12:25:03
 12:25:03.560944     8887.1          PID : 8887 Process : strmqtrc
 12:25:03.560974     8887.1          ------------------------------
 12:25:03.561057     8887.1     -{  TermPrivateServices
 12:25:03.561092     8887.1     --{  xxxTerminate
 12:25:03.561142     8887.1     ---{  xcsDestroyThreadMutexSem
 12:25:03.561203     8887.1     ---}  xcsDestroyThreadMutexSem rc=OK
 12:25:03.561227     8887.1     ---{  xcsFreeMem
 12:25:03.561346     8887.1          component:23 pointer:3d4e8
 12:25:03.561378     8887.1     ---}  xcsFreeMem rc=OK
 12:25:03.561413     8887.1     ---{  xcsFreeMem
 12:25:03.561445     8887.1          component:23 pointer:3f670
 12:25:03.561469     8887.1     ---}  xcsFreeMem rc=OK
 12:25:03.561491     8887.1     --}  xxxTerminate rc=OK
 12:25:03.561532     8887.1     --{  xehTerminateAsySignalHandling
 12:25:03.561567     8887.1     ---{  xehStopAsySignalMonitor
 12:25:03.561640     8887.1          xihAsySignalMonitorMutex holder
                                     tid(0) mod(0)
 12:25:03.561667     8887.1     ----{  xcsRequestThreadMutexSem
 12:25:03.561690     8887.1     ----}  xcsRequestThreadMutexSem rc=OK
 12:25:03.561714     8887.1     ----{  xcsFreeMem
 12:25:03.561745     8887.1          component:23 pointer:3c398
 12:25:03.561769     8887.1     ----}  xcsFreeMem rc=OK
 12:25:03.561791     8887.1     ----{  xcsFreeMem
 12:25:03.561820     8887.1          component:23 pointer:3c360
 12:25:03.561844     8887.1     ----}  xcsFreeMem rc=OK
 12:25:03.561948     8887.1     ----{  xcsFreeMem
 12:25:03.561980     8887.1          component:23 pointer:3c328
 12:25:03.562005     8887.1     ----}  xcsFreeMem rc=OK
 12:25:03.562026     8887.1     ----{  xcsFreeMem
 12:25:03.562055     8887.1          component:23 pointer:3b880
 12:25:03.562079     8887.1     ----}  xcsFreeMem rc=OK
 12:25:03.562101     8887.1     ----{  xcsReleaseThreadMutexSem
 12:25:03.562123     8887.1     ----}  xcsReleaseThreadMutexSem rc=OK
 12:25:03.562159     8887.1     ----{  xcsThreadRaise
 12:25:03.562192     8887.1          tid(2) sig(19)
 12:25:03.562219     8887.1          xcsThreadRaise: Before Lock
 12:25:03.562246     8887.1          xcsThreadRaise: After Lock
 12:25:03.562273     8887.1          top of loop
 12:25:03.562307     8887.1          In Loop: pCtl = 0x0003ae00
 12:25:03.562337     8887.1          calling pthread_kill for pCtl(3b8d8)
 12:25:03.562452     8887.1          thread kill ok
 12:25:03.562482     8887.1     ----}  xcsThreadRaise rc=OK
 12:25:03.562515     8887.1          Successfully raised SIGPWR for tid(2)
 12:25:03.562898     8887.1          Destroyed xihAsy
                                     Started condition variable
```

Figure 25. Sample WebSphere MQ for Solaris trace

Figure 26 on page 235 shows an extract from a WebSphere MQ for Linux for Intel and Linux for zSeries trace:

```
Timestamp        Process.Thread  Trace Data
========================================
 13:14:57.514231      406.1     --------}! xlsWaitEvent rc=xecL_W_TIMEOUT
 13:14:57.514250      406.1     -------}! xcsWaitEventSem rc=xecL_W_TIMEOUT
 13:14:57.514256      406.1     ------}! zcpReceiveOnPipe rc=zrcC_E_TIMEOUT
 13:14:57.514262      406.1     ------{  ziiHealthCheck
 13:14:57.514266      406.1     -------{  xcsCheckProcess
 13:14:57.514276      406.1          pid(390)
 13:14:57.514283      406.1     -------}  xcsCheckProcess rc=OK
 13:14:57.514286      406.1     -------{  xcsCheckProcess
 13:14:57.514293      406.1          pid(413)
 13:14:57.514298      406.1     -------}  xcsCheckProcess rc=OK
 13:14:57.514302      406.1     ------}  ziiHealthCheck rc=OK
 13:14:57.514306      406.1     ------{  zcpReceiveOnPipe
 13:14:57.514310      406.1     -------{  xcsWaitEventSem
 13:14:57.514315      406.1     --------{  xlsWaitEvent
 13:14:57.514323      406.1          Event(0x4090f724) Timeout(10000) Posted(0)
 13:14:57.514327      406.1     ---------{  xlsLockEvent
 13:14:57.514333      406.1     ---------}  xlsLockEvent rc=OK
 13:14:57.514336      406.1     ---------{  xlsUnlockEvent
 13:14:57.514341      406.1     ---------}  xlsUnlockEvent rc=OK
 13:14:57.514345      406.1     ---------{  xtmRequestCallback
 13:14:57.514350      406.1     ----------{  xtmStartTimerThread
 13:14:57.514354      406.1     -----------{  xcsCreateThread
 13:14:57.514365      406.1          Attr(260) pThread(0x401a5068) pArg(0)
 13:14:57.514374      406.1          pthread_mutex_lock ok
 13:14:57.514382      406.1          rejected non-pool thread (0x8078b40)
 13:14:57.514390      406.1          rejected poolThreadP(0x807a270)
                                          - !mayBeSleeping
 13:14:57.514421      406.25663      pthread_cond_timedwait completed
                                          poolThreadP(0x8082f30) rc(0)
 13:14:57.514437      406.25663      pool thread with poolThreadP(0x8082f30)
                                          redispatched for tid(25663)
 13:14:57.514443      406.25663 {  xppInitialiseDestructorRegistrations
 13:14:57.514451      406.25663      tP(0x8082f30)
 13:14:57.514455      406.25663 }  xppInitialiseDestructorRegistrations rc=OK
```

Figure 26. Sample WebSphere MQ for Linux for Intel and Linux for zSeries trace

Trace files

All trace files are created in the directory `/var/mqm/trace`.

Note: You can accommodate production of large trace files by mounting a temporary file system over this directory.

SSL trace files have the names AMQ.SSL.TRC and AMQ.SSL.TRC.1. You cannot format SSL trace files; send them unchanged to IBM support.

Non-SSL trace-file names have the following format:

`AMQ`*`ppppp`*`.TRC`

where *ppppp* is the process identifier (PID) of the process producing the trace.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

Tracing Secure Sockets Layer (SSL) on UNIX systems

On UNIX systems, you can request trace information for either iKeyman, the SSL functions, or both:

- To request iKeyman tracing, execute the gsk6ikm command with the –D flag:

  ```
  gsk6ikm -Dkeyman.debug=true
  ```

  iKeyman writes three trace files to the directory from which you start it, so consider starting iKeyman from the `/var/mqm/trace` directory. The trace files iKeyman generates are:

  ikmgdbg.log Java™ related trace

  ikmjdbg.log JNI related trace

  ikmcdbg.log C related trace

First-failure support technology (FFST)

This section describes the role of first-failure support technology (FFST) for WebSphere MQ.

FFST: WebSphere MQ for Windows

In WebSphere MQ for Windows, FFST information is recorded in a file in the `c:\Program Files\IBM\WebSphere MQ\errors` directory.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or a WebSphere MQ internal error.

FFST files are named `AMQ`*`nnnnn`*`.`*`mm`*`.FDC`, where:
nnnnn Is the ID of the process reporting the error
mm Is a sequence number, normally 0

When a process creates an FFST record it also sends a record to the Event Log. The record contains the name of the FFST file to assist in automatic problem tracking. The Event log entry is made at the application level.

A typical FFST log is shown in Figure 27 on page 237.

```
+------------------------------------------------------------------------------+
|                                                                              |
| WebSphere MQ First Failure Symptom Report                                    |
| =========================================                                    |
|                                                                              |
| Date/Time         :- Tue February 19 12:58:42 GMT Standard Time 2002         |
| Host Name         :- NETTLE (NT Version 4.0 Build 1381: Service Pack 6)      |
| PIDS              :- 5724B4101                                               |
| LVLS              :- 530                                                     |
| Product Long Name :- WebSphere MQ for Windows                                |
| Vendor            :- IBM                                                     |
| Probe Id          :- XC371019                                                |
| Application Name  :- MQM                                                     |
| Component         :- xstServerRequest                                        |
| Build Date        :- Feb 13 2002                                             |
| CMVC level        :- p000-L020213                                            |
| Build Type        :- IKAP - (Production)                                     |
| UserID            :- nigel                                                   |
| Process Name      :- C:\Program Files\IBM\WebSphere MQ\bin\amqzdmaa.exe       |
| Process           :- 00001678                                                |
| Thread            :- 00000001                                                |
| QueueManager      :- REGR                                                    |
| Major Errorcode   :- xecF_E_UNEXPECTED_SYSTEM_RC                             |
| Minor Errorcode   :- OK                                                      |
| Probe Type        :- MSGAMQ6119                                              |
| Probe Severity    :- 2                                                       |
| Probe Description :- AMQ6119: An internal WebSphere MQ error has occurred    |
|   (WinNT error 5 from WaitForSingleObject.)                                  |
| FDCSequenceNumber :- 0                                                       |
| Comment1          :- WinNT error 5 from WaitForSingleObject.                 |
|                                                                              |
| Comment2          :- Access is denied.                                       |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+

MQM Function Stack
amqzdmaa.main
xcsTerminate
xcsDisconnectSharedSubpool
xcsDettachSharedSubpool
xcsGetSetConnectCount
xstGetExtentConnectCount
xstStorageRequest
xstServerRequest
xcsFFST

MQM Trace History
------} zcpSendOnPipe rc=OK
------{ zcpReceiveOnPipe
-------{ xcsWaitEventSem

...
```

Figure 27. Sample WebSphere MQ for Windows First Failure Symptom Report

The Function Stack and Trace History are used by IBM to assist in problem determination. In most cases there is little that the system administrator can do when an FFST record is generated, apart from raising problems through the IBM Support Center.

FFST: WebSphere MQ for UNIX systems

For WebSphere MQ for UNIX systems, FFST information is recorded in a file in the `/var/mqm/errors` directory.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or a WebSphere MQ internal error.

The files are named AMQ*nnnnn.mm*.FDC, where:

nnnnn	Is the ID of the process reporting the error
mm	Is a sequence number, normally 0

When a process creates an FFST record, it also sends a record to syslog. The record contains the name of the FFST file to assist in automatic problem tracking.

The syslog entry is made at the *user.error* level. See the operating-system documentation about syslog.conf for information about configuring this.

Some typical FFST data is shown in Figure 28.

```
+------------------------------------------------------------------------------+
|                                                                              |
| WebSphere MQ First Failure Symptom Report                                    |
| =========================================                                    |
|                                                                              |
| Date/Time         :- Friday March 15 17:56:51 SGT 2002                       |
| Host Name         :- sunrts3 (SunOS 5.7)                                     |
| PIDS              :- 5724B4102                                               |
| LVLS              :- 530                                                     |
| Product Long Name :- WebSphere MQ for Sun Solaris                            |
| Vendor            :- IBM                                                     |
| Probe Id          :- RM161000                                                |
| Application Name  :- MQM                                                     |
| Component         :- rrmChangeClq                                            |
| Build Date        :- Mar 13 2002                                             |
| CMVC level        :- p000-L020312                                            |
| Build Type        :- IKAP - (Production)                                     |
| UserID            :- 00001001 (mqm)                                          |
| Program Name      :- amqrrmfa                                                |
| Process           :- 00019454                                                |
| Thread            :- 00000001                                                |
| QueueManager      :- REGR                                                    |
| Major Errorcode   :- rrcE_CLUS_COMMAND_ERROR                                 |
| Minor Errorcode   :- OK                                                      |
| Probe Type        :- MSGAMQ9413                                              |
| Probe Severity    :- 2                                                       |
| Probe Description :- AMQ9413: Repository command format error, command code |
|   0                                                                          |
| FDCSequenceNumber :- 0                                                       |
|                                                                              |
+------------------------------------------------------------------------------+

MQM Function Stack
rrmProcessMsg
rrmChangeClq
xcsFFST

MQM Trace History
---{ zstVerifyPCD
---} zstVerifyPCD rc=OK
---{ ziiMQCMIT
----{ ziiCreateIPCCMessage
-----{ zcpCreateMessage
-----} zcpCreateMessage rc=OK
----} ziiCreateIPCCMessage rc=OK
----{ ziiSendReceiveAgent
-----{ zcpSendOnPipe

...
```

Figure 28. FFST report for WebSphere MQ for UNIX systems

The Function Stack and Trace History are used by IBM to assist in problem determination. In most cases there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center.

However, there are some problems that the system administrator might be able to solve. If the FFST shows *out of resource* or *out of space on device* descriptions when calling one of the IPC functions (for example, **semop** or **shmget**), it is likely that the relevant kernel parameter limit has been exceeded.

If the FFST report shows a problem with **setitimer**, it is likely that a change to the kernel timer parameters is needed.

To resolve these problems, increase the IPC limits, rebuild the kernel, and restart the machine. See one of the following for further information:

- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*

Problem determination with clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are 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 might also be logged at the server, depending on the nature of the failure.

Terminating clients

Even though a client has terminated, the process at the server can still hold its queues open. Normally, this is only 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, WebSphere MQ attempts to place the error message in an error log in the root directory of the client machine.

UNIX systems clients

Error messages for UNIX clients are placed in the error logs in the same way as they are for the respective WebSphere MQ server systems. Typically these files appear in `/var/mqm/errors` on UNIX systems.

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 the C drive.

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.

Note: The default library does not contain the directory name for the qm.ini file because configuration information is stored in the Windows Registry.

The names of the default files held in this library are:

AMQERR01.LOG
For error messages.

AMQERR01.FDC
For First Failure Data Capture messages.

For more information about clients, see *WebSphere MQ Clients*.

Part 6. WebSphere MQ control commands

Chapter 16. How to use WebSphere MQ control commands

This chapter describes how to use the WebSphere MQ control commands. If you want to issue control commands, your user ID must be a member of the mqm group. For more information about this, see "Authority to administer WebSphere MQ" on page 113. In addition, note the following environment-specific information:

WebSphere MQ for Windows
All control commands can be issued from a command line. A subset can be issued using the WebSphere MQ Explorer snap-in. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case sensitive.

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

WebSphere MQ for UNIX systems
All WebSphere MQ control commands can be issued from a shell. All commands are case-sensitive.

Names of WebSphere MQ objects

In general, the names of WebSphere MQ objects can have up to 48 characters. This rule applies to all the following objects:
- Queue managers
- Queues
- Process definitions
- Namelists
- Clusters
- Authentication information objects

The maximum length of channel names is 20 characters.

The characters that can be used for all WebSphere MQ names are:
- Uppercase A–Z
- Lowercase a–z
- Numerics 0–9
- Period (.)
- Underscore (_)
- Forward slash (/) (see note 1)
- Percent sign (%) (see note 1)

Notes:

1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.
2. Leading or embedded blanks are not allowed.
3. National language characters are not allowed.
4. Names can be enclosed in double quotation marks, but this is essential only if special characters are included in the name.

How to read syntax diagrams

This book contains syntax diagrams (sometimes referred to as *railroad* diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

Table 20. How to read syntax diagrams

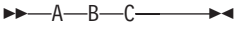
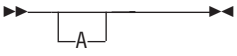
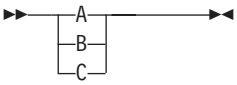
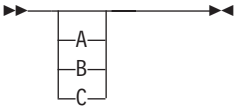
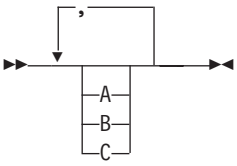
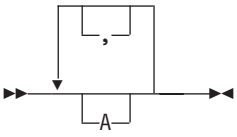
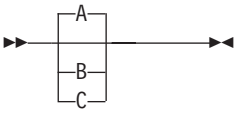
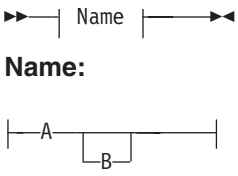
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
	You can specify value A. Optional values are shown below the main line of a syntax diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and C are alternatives, one of which you might specify.
	You can specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You can specify value A multiple times. The separator in this example is optional.
	Values A, B, and C are alternatives, one of which you can specify. If you specify none of the values shown, the default A (the value shown above the main line) is used.

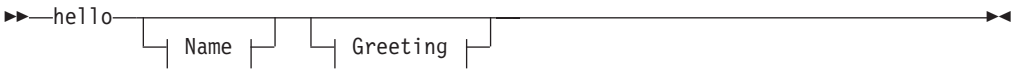
Table 20. How to read syntax diagrams (continued)

Convention	Meaning
	The syntax fragment `Name` is shown separately from the main syntax diagram.
Punctuation and uppercase values	Specify exactly as shown.

Example syntax diagram

Here is an example syntax diagram that describes the **hello** command:

Hello Command



Name

Greeting

Notes:

1 You can code up to three names.

According to the syntax diagram, these are all valid versions of the **hello** command:

```
hello
hello name
hello name, name
hello name, name, name
hello, how are you?
hello name, how are you?
hello name, name, how are you?
hello name, name, name, how are you?
```

The space before the *name* value is significant, and that if you do not code *name* at all, you must still code the comma before `how are you?`.

Syntax help

You can obtain help for the syntax of any control command by entering the command followed by a question mark. WebSphere MQ responds by listing the syntax required for the selected command.

The syntax shows all the parameters and variables associated with the command. Different forms of parentheses are used to indicate whether a parameter is required. For example:

```
CmdName [-x OptParam ] ( -c | -b ) argument
```

where:

CmdName
Is the command name for which you have requested help.

[-x OptParam]
Square brackets enclose one or more optional parameters. Where square brackets enclose multiple parameters, you can select no more than one of them.

(-c | -b)
Brackets enclose multiple values, one of which you must select. In this example, you must select either flag c or flag b.

argument
A mandatory argument.

Examples

1. Result of entering `endmqm ?`

   ```
   endmqm [-z][-c | -w | -i | -p] QMgrName
   ```

2. Result of entering `rcdmqimg ?`

   ```
   rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]
   ```

Chapter 17. The control commands

This section provides reference information for each of the following WebSphere MQ control commands:

Command name	Purpose
amqmcert	Manage certificates for SSL
amqmdain	Configure or control WebSphere MQ services (Windows systems only)
crtmqcvx	Convert data
crtmqm	Create a local queue manager
dltmqm	Delete a queue manager
dmpmqaut	Dump authorizations to an object
dmpmqlog	Dump a log
dspmq	Display queue managers
dspmqaut	Display authorizations to an object
dmpmqcap	Display processor capacity and number of processors
dspmqcsv	Display the status of a command server
dspmqfls	Display file names
dspmqtrc	Display formatted trace output (HP-UX, Linux, and Solaris)
dspmqrtn	Display details of transactions
endmqcsv	Stop the command server on a queue manager
endmqlsr	Stop the listener process on a queue manager
endmqm	Stop a local queue manager
endmqtrc	Stop tracing for an entity (not for AIX)
rcdmqimg	Write an image of an object to the log
rcrmqobj	Recreate an object from their image in the log
rsvmqtrn	Commit or back out a transaction
runmqchi	Start a channel initiator process
runmqchl	Start a sender or requester channel
runmqdlq	Start the dead-letter queue handler
runmqlsr	Start a listener process
runmqsc	Issue MQSC commands to a queue manager
runmqtmc	Invoke a trigger monitor for a client (AIX clients only)
runmqtrm	Invoke a trigger monitor for a server
setmqaut	Change authorizations to an object
setmqcap	Set processor capacity
setmqcrl	Set certificate revocation list (CRL) LDAP server definitions (Windows systems only)
setmqscp	Set service connection points (Windows systems only)
strmqcsv	Start the command server for a queue manager
strmqm	Start a local queue manager

strmqtrc	Enable tracing (not for AIX)

amqmcert (manage certificates)

Purpose

Use the **amqmcert** command to manage certificates with WebSphere MQ for Windows. You can use this command to configure a WebSphere MQ client certificate store for SSL, or a WebSphere MQ queue manager certificate store. You can assign only personal certificates, which have associated private keys, to a queue manager or WebSphere MQ client. The local store must also contain CA certificates from the Certification Authorities that issue the certificates you expect to receive. CA certificates, also known as signer certificates, enable you to validate incoming certificates.

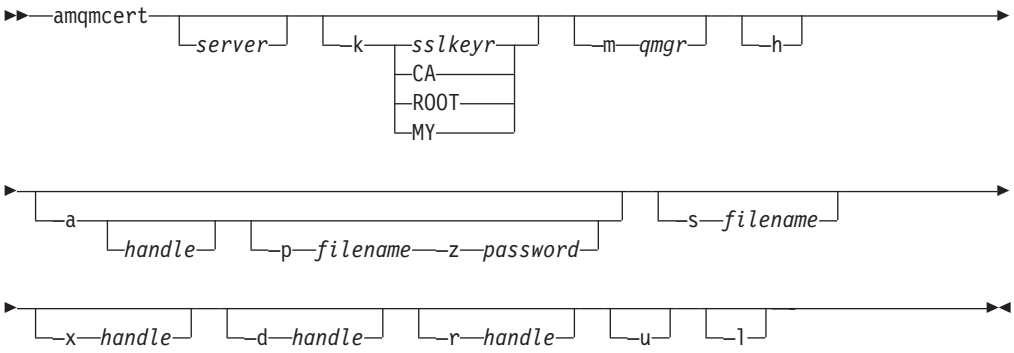
Usually, each queue manager and WebSphere MQ client has its own certificate store. If you want to change the location of a certificate store, refer to the information on working with your key repository in *WebSphere MQ Security*.

The **amqmcert** command enables you to copy certificates between certificate stores on a computer, or to import certificates from a file. You can copy and install private key data if you require.

Notes:

1. Only administrators or members of the mqm group can use amqmcert to list or modify WebSphere MQ queue manager stores. General users can use it to list a client or system store or modify a client store.
2. To manage certificates on UNIX systems, you use IKEYCMD from the command line, or the iKeyman GUI. Chapter 18, "Using the IKEYCMD interface to manage keys and certificates on UNIX systems" on page 319describes the command line interface.

Syntax



Optional parameters

server
The name of the machine at which the command is targeted. For example, you can use the **amqmcert** command to copy certificates from the interactive user's personal certificate store to a queue manager store on a remote computer on the network. If you include this parameter, it must precede all other options. If you omit it, the command is executed on the local machine.

-k *sslkeyr* | **CA** | **ROOT** | **MY**
Either:

- An alternative MQSSLKEYR value for the current operation when you are manipulating WebSphere MQ client stores, or
- The source system certificate store for a copy or enumerate operation. CA, ROOT, and MY are names given to system certificate stores provided by Microsoft Internet Explorer or Windows 2000.

-m *qmgr*
The target queue manager. Operations look up the queue manager's SSLKEYR value to locate the certificate store and use that as the target.

-h Specifies that the command refers to the local machine's certificate stores. Windows systems allow two sets of certificate stores, which reside in the registry. One is based on the currently logged in user (HKEY_CURRENT_USER), and the other is for all users of the local machine (HKEY_LOCAL_MACHINE). By default, when using the -k parameter, the command refers to the current user's certificate stores.

-a *handle*
Adds a certificate to a store. When you specify a *handle*, the command copies the certificate identified by *handle* to the store. If the certificate being copied has associated private key data, that data is also copied to the local machine's private key store, if that data does not already exist.

-p *filename*
The filename for a source personal certificate. These files usually (but not always) contain private key data, and require a password (provided by the -z option) to enable decryption and import. Personal certificates can be assigned to a WebSphere MQ queue manager or WebSphere MQ client. Personal certificate files are usually provided in files with extensions .p12 or .pfx.

Note that this function is available only on Windows 2000 or later versions.

-z *password*
The password required to decrypt and import a personal certificate.

-s *filename*
The filename for a CA certificate, used when verifying a subject certificate. CA certificates are not usually encrypted, and typically are provided in files with extensions .DER, .pb7, or .CER.

-x *handle*
Exports the certificate identified by *handle* to a file, specified with the -s option. The file is exported in a proprietary format, and can only be used by this command to import to another WebSphere MQ store. Private key data is also exported if a password has been supplied and the private key data exists.

-d *handle*
Assigns the certificate identified by *handle* to the target queue manager or client. When you use the -m option to specify a queue manager, the certificate must exist in that queue manager's store, and is assigned as the certificate to use to identify that queue manager to remote parties (WebSphere MQ clients). Without the -m option, the certificate is assigned to the current WebSphere MQ client, that is, the logged- in user.

-r*handle*
Removes the certificate identified by *handle* from the store.

-u Unassigns a certificate from a queue manager, when the -m option is specified, or from the current WebSphere MQ client, that is, the logged-in user.

-l Lists certificates. Use this option to enumerate the available certificates. The context of the enumeration can be specified using the -m or -k options, to

specify either a queue manager store, a system store, or a specific client store. If neither -m nor -k are specified, the command lists the client store targeted by the environment variable MQSSLKEYR. Certificates are listed with unique identifiers (handles), which can then be used on subsequent commands to identify source or target certificates.

Examples

For a WebSphere MQ client, ensure that the MQSSLKEYR environment variable is set to the root filename of the client certificate store. For example, when you set MQSSLKEYR=D:\mqm\key the **amqmcert** command uses or creates a store named D:\mqm\key.sto.

Enumerating certificate stores

amqmcert -l

Lists the contents of the store referred to by MQSSLKEYR, that is, the current user's WebSphere MQ client store.

amqmcert -l -k d:\mqm\key

Lists the contents of the store d:\mqm\key.sto.

amqmcert -l -m QM1

Lists the contents of the queue manager store for queue manager QM1.

amqmcert -l -k ROOT -h

Lists all certificates in the local machine ROOT store that are available to the local machine.

amqmcert -l -k CA -h

Lists all certificates in the local machine CA store that are available to the local machine.

amqmcert -l -k MY

Lists all certificates in the current user's MY system store.

Copying a certificate from one store to another

amqmcert -m QM2 -a 102

Copies the certificate with handle 102 into the queue manager store for QM2.

amqmcert -a 102

Copies the certificate whose handle is 102 to the WebSphere MQ store for the current user.

amqmcert -a 4 -k MY -m QM1

Copies the certificate with handle 4 from the current user's MY system store to the queue manager store for QM1. If the certificate has private key data, that data will be copied to the local machine's private key repository.

Assigning a certificate for use by WebSphere MQ

amqmcert -m QM2 -d 122

Assigns the certificate with handle 122 to be used for authenticating QM2. Note that the certificate must be in the queue manager store.

amqmcert -d 123

Assigns the certificate with handle 123 to be used to authenticate this WebSphere MQ client, that is, the interactive user.

Importing a certificate from a file

Note that this function is available only on Windows 2000 and Windows XP.

Importing personal certificates: Personal certificates are usually provided in files with extensions `.p12` or `.pfx`:

amqmcert -a -p mqper.pfx -z password
Imports the certificates in `mqper.pfx` to the current user's WebSphere MQ client store using the specified password to decrypt the private key, which is then stored in the local machine's key repository.

Importing CA certificates: CA certificates are usually provided in files with the extension `.cer`:

amqmcert -a -s mqcacert.cer
Imports the certificates in `mqcacert.cer` to the current user's WebSphere MQ client store.

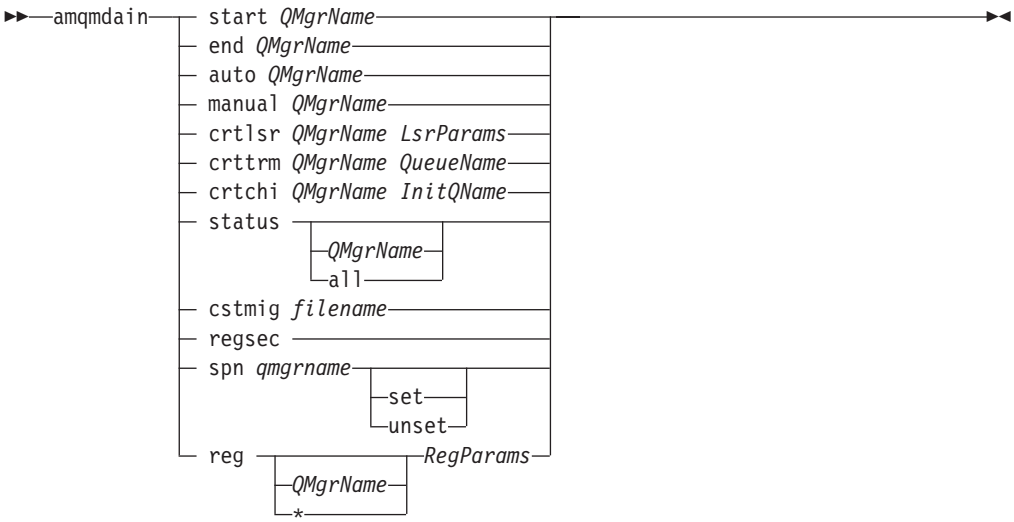
amqmdain (WebSphere MQ services control)

Purpose

The amqmdain command applies to WebSphere MQ for Windows only.

Use **amqmdain** to configure or control WebSphere MQ Services, as an alternative to using the WebSphere MQ Services snap-in. Starting a queue manager service with **amqmdain** is not the same as using **strmqm** from the command line, because WebSphere MQ Services execute in a non-interactive session, running under a different user account. You can also configure a queue manager service to start associated processes, such as listeners and trigger monitors, or to ensure that any registry entries you have edited manually are assigned the correct security permissions.

Syntax



Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager to which the command applies.

start *QMgrName*
Starts a queue manager service.

end *QMgrName*
Ends a queue manager service.

auto *QMgrName*
Sets a queue manager service to automatic startup.

manual *QMgrName*
Sets a queue manager service to manual startup.

crtlsr *QMgrName LsrParams*
Creates a listener service for a queue manager.

LsrParams
Parameters applicable to the **runmqlsr** command, for example `-t trptype -p Port`. The parameters must be in pairs, but the `-m` *QMgrName* parameter is not required, because it is specified by the preceding parameter to the **crtlsr** keyword. See "runmqlsr (run listener)" on page 295 for a complete description of the **runmqlsr** command.

crttrm *QMgrName QueueName*
Creates a trigger monitor service for a queue manager.

QueueName
The name of the queue to be used by the trigger monitor service. See "runmqtmc (start client trigger monitor)" on page 300 for a complete description of the **runmqtrm** command.

crtchi *QMgrName InitQName*
Creates a channel initiator service for a queue manager.

InitQName
The name of the initiation queue to be used by the channel initiator. See "runmqchl (run channel)" on page 293 for a complete description of the **runmqchi** command.

status *QMgrName* **| all**
These parameters are optional.

If no parameter is supplied:	Displays the status of the WebSphere MQ service.
If a *QMgrName* is supplied:	Displays the status of the named queue manager service.
If the parameter *all* is supplied:	Displays the status of all services.

cstmig *filename*
Imports definitions of custom services.

amqmdain loads custom services from a comma-separated value configuration file. **amqmdain must** be executed to store the custom service parameters in the Registry, add the key and values in the correct place, and to assign the appropriate security permissions to the Registry.

The format of an entry in the configuration file is:

```
Command Name, Start Command, End Command, Flags, Reserved
```

For example:

```
PubSub Broker, strmqbrk -p blue.queue.manager, endmqbrk -i
  -m blue.queue.manager, SUFFIX|ROOT|STARTUP|SHUTDOWN|COMMAND, 1
```

regsec
Ensures that the security permissions assigned to the Registry keys are correct.

spn *QMgrName* **set | unset**
Allows you to set or unset the service principal name for a queue manager.

reg *QMgrName* **| *** *RegParams*
Modifies some WebSphere MQ attributes in the Windows Registry.

RegParams specifies the Registry stanzas to change and the changes to make. It takes one of the following forms:

```
-c add -s stanza -v attribute=value
-c remove -s stanza -v [attribute|*]
-c display -s stanza -v [attribute|*]
```

When you specify *, the valid values for stanza are:

```
ApiExitCommon\name
ApiExitTemplate\name
ACPI
AllQueueManagers
Channels
DefaultQueueManager
LogDefaults
```

When you specify a particular queue manager, the valid values for stanza are:

```
XAResourceManager\name
ApiExitLocal\name
Channels
ExitPath
Log
QueueManagerStartup
TCP
LU62
SPX
NetBios
```

amqmdain does not validate any name, attribute, or value you specify.

When you specify add, and an attribute already exists, it is modified. If a stanza does not exist, **amqmdain** creates it.

When you specify remove or display, you can use the value * to remove or display all attributes. If you use remove to delete the last attribute in a stanza, the stanza itself is deleted.

Any modification you make to the Registry re-secures all WebSphere MQ Registry entries.

Examples

The following example adds an XAResourceManager to queue manager TEST. The commands issued are:

```
amqmdain reg TEST -c add -s XAResourceManager\Sample -v SwitchFile=sf1
amqmdain reg TEST -c add -s XAResourceManager\Sample -v ThreadOfControl=THREAD
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XAOpenString=openit
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XACloseString=closeit
```

To display the values set by the commands above, use:

```
amqmdain reg TEST -c display -s XAResourceManager\Sample -v *
```

The display would look something like this:

```
0784726, 5639-B43 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Displaying registry value for Queue Manager 'TEST'
        Attribute = Name, Value = Sample
        Attribute = SwitchFile, Value = sf1
        Attribute = ThreadOfControl, Value = THREAD
        Attribute = XAOpenString, Value = openit
        Attribute = XACloseString, Value = closeit
```

To remove the XAResourceManager from queue manager TEST, use:

```
amqmdain reg TEST -c remove -s XAResourceManager\Sample -v *
```

Return codes

0	Command completed normally
-2	Syntax error
-3	Failed to initialize COM library
-4	Failed to initialize COM components
-5	Failed to create channel initiator
-6	Failed to create service
-7	Failed to configure service
-8	Invalid port type specified for crtlsr
-9	Unexpected Registry error

crtmqcvx (data conversion)

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert C structures.

The command reads an input file containing structures to be converted, and writes an output file containing code fragments to convert those structures.

For information about using this command, see the *WebSphere MQ Application Programming Guide*.

Syntax

Required parameters

SourceFile
The input file containing the C structures to convert.

TargetFile
The output file containing the code fragments generated to convert the structures.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source.tmp target.c
```

The input file, `source.tmp` looks like this:

```
/* This is a test C structure which can be converted by the */
/* crtmqcvx utility                                         */

struct my_structure
{
    int    code;
    MQLONG value;
};
```

The output file, target.c, produced by the command is shown below. You can use these code fragments in your applications to convert data structures. However, if you do so, the fragment uses macros supplied in the header file amqsvmha.h.

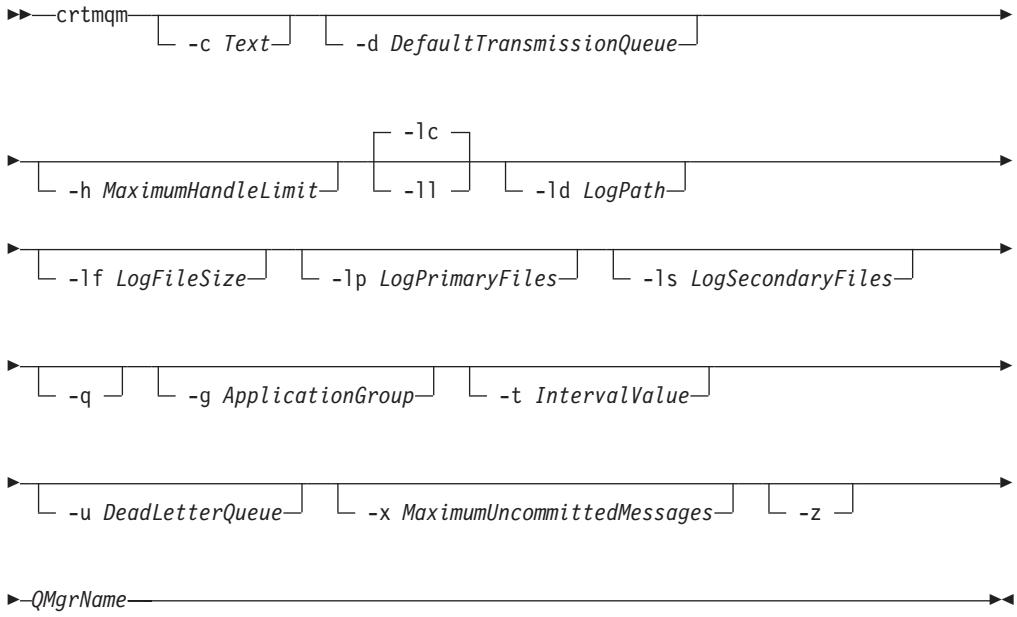
```
MQLONG Convertmy_structure(
             PMQBYTE *in_cursor,
             PMQBYTE *out_cursor,
             PMQBYTE in_lastbyte,
             PMQBYTE out_lastbyte,
             MQHCONN hConn,
             MQLONG  opts,
             MQLONG  MsgEncoding,
             MQLONG  ReqEncoding,
             MQLONG  MsgCCSID,
             MQLONG  ReqCCSID,
             MQLONG  CompCode,
             MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}
```

crtmqm (create queue manager)

Purpose

Use the **crtmqm** command to create a local queue manager and define the default and system objects. The objects created by **crtmqm** are listed in Appendix A, "System and default objects" on page 475. When a queue manager has been created, use the **strmqm** command to start it.

Syntax



Required parameters

QMgrName
The name of the queue manager to create. The name can contain up to 48 characters. This must be the last item in the command.

Optional parameters

-c *Text*
Descriptive text for this queue manager. You can use up to 64 characters; the default is all blanks.

If you include special characters, enclose the description in double quotes. The maximum number of characters is reduced if the system is using a double-byte character set (DBCS).

-d *DefaultTransmissionQueue*
The name of the local transmission queue where remote messages are put if a transmission queue is not explicitly defined for their destination. There is no default.

-h *MaximumHandleLimit*
The maximum number of handles that any one application can have open at the same time.

Specify a value in the range 1 through 999 999 999. The default value is 256.

The next six parameter descriptions relate to logging, which is described in "Using the log for recovery" on page 204.

Note: Choose the logging arrangements with care, because you cannot change them once they are committed.

-lc Use circular logging. This is the default logging method.

-ll Use linear logging.

-ld *LogPath*
The directory used to hold log files.

In WebSphere MQ for Windows, the default is `C:\Program Files\IBM\WebSphere MQ\log\qmgr` (assuming that C is your data drive).

In WebSphere MQ for UNIX systems, the default is `/var/mqm/log`.

User ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This occurs automatically if the log files are in their default locations.

-lf *LogFileSize*
The size of the log files in units of 4 KB.

In WebSphere MQ for Windows, the minimum value is 32, and the maximum is 16 384. The default value is 256, giving a default log size of 1 MB.

In WebSphere MQ for UNIX systems, the minimum value is 64, and the maximum is 16 384. The default value is 1024, giving a default log size of 4 MB.

-lp *LogPrimaryFiles*
The number of primary log files to be allocated. The default value is 3, the minimum is 2, and the maximum is 62.

-ls *LogSecondaryFiles*
The number of secondary log files to be allocated. The default value is 2, the minimum is 1, and the maximum is 61.

Note: The total number of log files is restricted to 63, regardless of the number requested.

The limits given in the previous parameter descriptions are limits set by WebSphere MQ. Operating system limits might reduce the maximum possible log size.

-q Makes this queue manager the default queue manager. The new queue manager replaces any existing default queue manager.

If you accidentally use this flag and want to revert to an existing queue manager as the default queue manager, change the default queue manager as described in "Making an existing queue manager the default" on page 28.

-g *ApplicationGroup*
The name of the group containing members allowed to:
- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories

This option applies only to WebSphere MQ for AIX, Solaris, HP-UX, and Linux.

The default value is -g all, which allows unrestricted access.

The -g *ApplicationGroup* value is recorded in the queue manager configuration file, qm.ini.

The mqm user ID **must** belong to the specified ApplicationGroup.

-t *IntervalValue*
The trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after receiving a trigger-generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You might choose to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, an application can enable triggering again by servicing the queue using a command to alter the queue to reset the trigger attribute.

-u *DeadLetterQueue*
The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default is no dead-letter queue.

-x *MaximumUncommittedMessages*
The maximum number of uncommitted messages under any one syncpoint. That is, the sum of:
- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside a syncpoint.

Specify a value in the range 1 through 999 999 999. The default value is 10 000 uncommitted messages.

-z Suppresses error messages.

This flag is used within WebSphere MQ to suppress unwanted error messages. Because using this flag can result in loss of information, do not use it when entering commands on a command line.

Return codes

0	Queue manager created
8	Queue manager already exists
49	Queue manager stopping

69	Storage not available
70	Queue space not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid
111	Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
115	Invalid log size

Examples

1. This command creates a default queue manager called `Paint.queue.manager`, with a description of `Paint shop`, and creates the system and default objects. It also specifies that linear logging is to be used:

   ```
   crtmqm -c "Paint shop" -ll -q Paint.queue.manager
   ```

2. This command creates a default queue manager called `Paint.queue.manager`, creates the system and default objects, and requests two primary and three secondary log files:

   ```
   crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
   ```

3. This command creates a queue manager called `travel`, creates the system and default objects, sets the trigger interval to 5000 milliseconds (or 5 seconds), and specifies SYSTEM.DEAD.LETTER.QUEUE as its dead-letter queue.

   ```
   crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
   ```

Related commands

strmqm	Start queue manager
endmqm	End queue manager
dltmqm	Delete queue manager

dltmqm (delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager and all objects associated with it. Before you can delete a queue manager you must end it using the **endmqm** command.

In WebSphere MQ for Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Syntax

Required parameters

QMgrName
The name of the queue manager to delete.

Optional parameters

-z Suppresses error messages.

Return codes

0	Queue manager deleted
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
24	A process that was using the previous instance of the queue manager has not yet disconnected.
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid
112	Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.

Examples

1. The following command deletes the queue manager `saturn.queue.manager`.
   ```
   dltmqm saturn.queue.manager
   ```
2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command.
   ```
   dltmqm -z travel
   ```

Related commands

crtmqm	Create queue manager

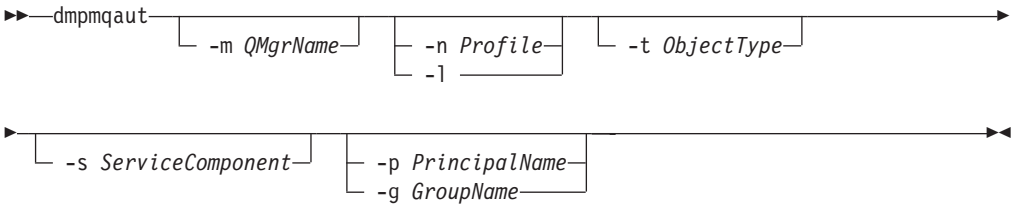
strmqm	Start queue manager
endmqm	End queue manager

dmpmqaut (dump authority)

Purpose

Use the **dmpmqaut** command to dump the current authorizations to a specified object.

Syntax



Optional parameters

-m *QMgrName*
Dump authority records only for the queue manager specified. If you omit this parameter, only authority records for the default queue manager are dumped.

-n *Profile*
The name of the profile for which to dump authorizations. The profile name can be generic, using wildcard characters to specify a range of names as explained in "Using OAM generic profiles" on page 123.

-l Dump only the profile name and type. Use this option to generate a *terse* list of all defined profile names and types.

-t *ObjectType*
The type of object for which to dump authorizations. Possible values are:

queue or **q**	A queue or queues matching the object name parameter
qmgr	A queue manager object
process or **prcs**	A process
namelist or **nl**	A namelist
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service for which to dump authorizations. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

-p *PrincipalName*
This parameter applies to WebSphere MQ for Windows only; UNIX systems keep only group authority records.

The name of a user for whom to dump authorizations to the specified object. The name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see “Principals and groups” on page 116.

-g *GroupName*
The name of the user group for which to dump authorizations. You can specify only one name, which must be the name of an existing user group. On Windows systems, you can use only local groups.

Examples

The following examples show the use of dmpmqaut to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

```
dmpmqaut -m qm1 -n a.b.c -t q -p user1
```

   The resulting dump would look something like this:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

   Note: UNIX users cannot use the `-p` option; they must use `-g groupname` instead.

2. This example dumps all authority records with a profile that matches queue a.b.c.

```
dmpmqaut -m qmgr1 -n a.b.c -t q
```

   The resulting dump would look something like this:

```
profile:     a.b.c
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - -
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
- - - - - - - - - - - - - - - -
profile:     a.**
object type: queue
entity:      group1
type:        group
authority:   get
```

3. This example dumps all authority records for profile a.b.*, of type queue.

```
dmpmqaut -m qmgr1 -n a.b.* -t q
```

   The resulting dump would look something like this:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

The resulting dump would look something like this:

```
profile:     q1
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

```
dmpmqaut -m qmX -l
```

The resulting dump would look something like this:

```
profile: q1, type: queue
profile: q*, type: queue
profile: name.*, type: namelist
profile: pr1, type: process
```

Note: For WebSphere MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

Related commands

dspmqaut	Display authority
setmqaut	Set or reset authority

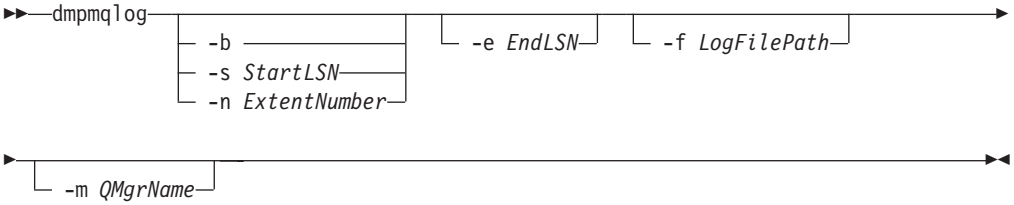
dmpmqlog (dump log)

Purpose

Use the **dmpmqlog** command to dump a formatted version of the WebSphere MQ system log.

The log to be dumped must have been created on the same type of operating system as that being used to issue the command.

Syntax



Optional parameters

Dump start point
Use one of the following parameters to specify the log sequence number (LSN) at which the dump should start. If you omit this, dumping starts by default from the LSN of the first record in the active portion of the log.

-b Start dumping from the base LSN. The base LSN identifies the start of the log extent that contains the start of the active portion of the log.

-s *StartLSN*
Start dumping from the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

If you are using a circular log, the LSN value must be equal to or greater than the base LSN value of the log.

-n *ExtentNumber*
Start dumping from the specified extent number. The extent number must be in the range 0–9 999 999.

This parameter is valid only for queue managers using linear logging.

-e *EndLSN*
End dumping at the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

-f *LogFilePath*
The absolute (rather than relative) directory path name to the log files. The specified directory must contain the log header file (`amqhlctl.lfh`) and a subdirectory called `active`. The active subdirectory must contain the log files. By default, log files are assumed to be in the directories specified in the WebSphere MQ configuration information. If you use this option, queue names associated with queue identifiers are shown in the dump only if you use the -m option to name a queue manager name that has the object catalog file in its directory path.

On a system that supports long file names this file is called qmqmobjcat and, to map the queue identifiers to queue names, it must be the file used when the

log files were created. For example, for a queue manager named qm1, the object catalog file is located in the directory `..\qmgrs\qm1\qmanager\`. To achieve this mapping, you might need to create a temporary queue manager, for example named tmpq, replace its object catalog with the one associated with the specific log files, and then start dmpmqlog, specifying `-m tmpq` and `-f` with the absolute directory path name to the log files.

-m *QMgrName*
The name of the queue manager. If you omit this parameter, the name of the default queue manager is used.

The queue manager must not be running when the **dmpmqlog** command is issued. Similarly, the queue manager must not be started while **dmpmqlog** is running.

dspmq (display queue managers)

Purpose

Use the **dspmq** command to display the names and operational status of all the queue managers on a machine.

Syntax

Required parameters

None

Optional parameters

-m *QMgrName*
The queue manager for which to display details. If you give no name, all queue manager names are displayed.

-s Requests the operational status of the queue managers.

Return codes

0	Command completed normally
36	Invalid arguments supplied
71	Unexpected error
72	Queue manager name error

dspmqaut (display authority)

Purpose

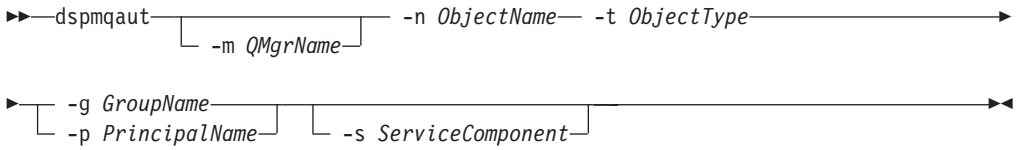
Use the **dspmqaut** command to display the current authorizations to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all the groups.

Only one group or principal can be specified.

For more information about authorization service components, see "Installable services" on page 100, "Service components" on page 101, and Chapter 20, "Authorization service" on page 341.

Syntax



Required parameters

-n *ObjectName*
The name of a queue manager, queue, or process definition on which to make the inquiry.

You must include this parameter, ***unless*** the inquiry relates to the queue manager itself, in which case you must omit it.

-t *ObjectType*
The type of object on which to make the inquiry. Possible values are:

queue or **q**	A queue or queues matching the object name parameter
qmgr	A queue manager object
process or **prcs**	A process
namelist or **nl**	A namelist
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security

Optional parameters

-m *QMgrName*
The name of the queue manager on which to make the inquiry. This parameter is optional if you are setting the authorizations of your default queue manager.

-g *GroupName*
The name of the user group on which to make the inquiry. You can specify only one name, which must be the name of an existing user group. On Windows systems, you can use only local groups.

-p *PrincipalName*
The name of a user for whom to display authorizations to the specified object.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see "Principals and groups" on page 116.

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

Returned parameters

Returns an authorization list, which can contain none, one, or more authorization values. Each authorization value returned means that any user ID in the specified group or principal has the authority to perform the operation defined by that value.

Table 21 shows the authorities that can be given to the different object types.

Table 21. Security authorities from the dspmqaut command

Authority	Queue	Process	Queue manager	Namelist	Authent-ication information
all	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No
browse	Yes	No	No	No	No
chg	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No
connect	No	No	Yes	No	No
crt	Yes	Yes	Yes	Yes	Yes
dlt	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No
inq	Yes	Yes	Yes	Yes	Yes
passall	Yes	No	No	No	No
passid	Yes	No	No	No	No
put	Yes	No	No	No	No
set	Yes	Yes	Yes	No	Yes
setall	Yes	No	Yes	No	No
setid	Yes	No	Yes	No	No

The following list defines the authorizations associated with each value:

all	Use all operations relevant to the object.
alladm	Perform all administration operations relevant to the object.
allmqi	Use all MQI calls relevant to the object.
altusr	Specify an alternate user ID on an MQI call.
browse	Retrieve a message from a queue by issuing an **MQGET** call with the BROWSE option.
chg	Change the attributes of the specified object, using the appropriate command set.
clr	Clear a queue (PCF command Clear queue only).
connect	Connect the application to the specified queue manager by issuing an **MQCONN** call.
crt	Create objects of the specified type using the appropriate command set.
dlt	Delete the specified object using the appropriate command set.
dsp	Display the attributes of the specified object using the appropriate command set.
get	Retrieve a message from a queue by issuing an **MQGET** call.
inq	Make an inquiry on a specific queue by issuing an **MQINQ** call.
passall	Pass all context.
passid	Pass the identity context.
put	Put a message on a specific queue by issuing an **MQPUT** call.
set	Set attributes on a queue from the MQI by issuing an **MQSET** call.
setall	Set all context on a queue.
setid	Set the identity context on a queue.

The authorizations for administration operations, where supported, apply to these command sets:
- Control commands
- MQSC commands
- PCF commands

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing

Examples

- The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

  ```
  dspmqaut -m saturn.queue.manager -t qmgr -g staff
  ```

  The results from this command are:

```
Entity staff has the following authorizations for object:
        get
        browse
        put
        inq
        set
        connect
        altusr
        passid
        passall
        setid
```

- The following example displays the authorities user1 has for queue a.b.c:

  ```
  dspmqaut -m qmgr1 -n a.b.c -t q -p user1
  ```

  The results from this command are:

  ```
  Entity user1 has the following authorizations for object:
          get
          put
  ```

Related commands

dmpmqaut	Dump authority
setmqaut	Set or reset authority

dspmqcap (display license units)

Purpose

Use the **dspmqcap** command to display the number of processors for which you have purchased license units.

To fulfil the conditions of your license agreement with IBM, you must have purchased sufficient license units for the number of processors on which you are running WebSphere MQ. The installation dialog checks for this when you install WebSphere MQ, and you can set it using the **setmqcap** command.

Syntax

Required parameters

None

Return codes

0	Successful operation
36	Invalid argument
71	Unexpected error

Related commands

setmqcap	Set license units

dspmqcsv (display command server)

Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

The status can be one of the following:
- Starting
- Running
- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets
- Ending
- Stopped

Syntax

Required parameters

None

Optional parameters

QMgrName
The name of the local queue manager for which the command server status is being requested.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command displays the status of the command server associated with `venus.q.mgr`:

```
dspmqcsv venus.q.mgr
```

Related commands

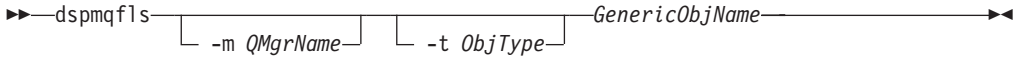
strmqcsv	Start a command server
endmqcsv	End a command server

dspmqfls (display files)

Purpose

Use the **dspmqfls** command to display the real file system name for all WebSphere MQ objects that match a specified criterion. You can use this command to identify the files associated with a particular object. This is useful for backing up specific objects. See "Understanding WebSphere MQ file names" on page 18 for information about name transformation.

Syntax



Required parameters

GenericObjName
The name of the object. The name is a string with no flag and is a required parameter. Omitting the name returns an error.

This parameter supports a wild card character * at the end of the string.

Optional parameters

-m *QMgrName*
The name of the queue manager for which to examine files. If you omit this name, the command operates on the default queue manager.

-t *ObjType*
The object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

* or **all**	All object types; this is the default
q or **queue**	A queue or queues matching the object name parameter
ql or **qlocal**	A local queue
qa or **qalias**	An alias queue
qr or **qremote**	A remote queue
qm or **qmodel**	A model queue
qmgr	A queue manager object
prcs or **process**	A process
ctlg or **catalog**	An object catalog
nl or **namelist**	A namelist
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security

Notes:

1. The **dspmqfls** command displays the name of the directory containing the queue, ***not*** the name of the queue itself.
2. In WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. The way you do

this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

Return codes

0	Command completed normally
10	Command completed but not entirely as expected
20	An error occurred during processing

Examples

1. The following command displays the details of all objects with names beginning SYSTEM.ADMIN defined on the default queue manager.

   ```
   dspmqfls SYSTEM.ADMIN*
   ```

2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

   ```
   dspmqfls -m RADIUS -t prcs PROC*
   ```

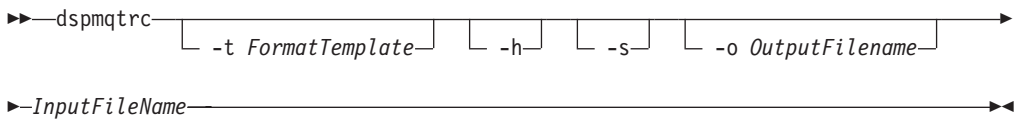
dspmqtrc (display formatted trace output)

Purpose

The dspmqtrc command is supported on Solaris, HP-UX, and Linux only.

Use the **dspmqtrc** command to display WebSphere MQ formatted trace output.

Syntax



Required parameters

InputFileName
The name of the file containing the unformatted trace. For example `/var/mqm/trace/AMQ12345.TRC`. If you provide one input file, **dspmqtrc** formats it either to stdout or to the output file you name. If you provide more than one input file, any output file you name is ignored, and formatted files are named `AMQXXXXX.FMT`, based on the PID of the trace file.

Optional parameters

-t *FormatTemplate*
The name of the template file containing details of how to display the trace. The default value is `/opt/mqm/lib/amqtrc.fmt`.

-h Omit header information from the report.

-s Extract trace header and put to stdout.

-o *output_filename*
The name of the file into which to write formatted data.

Related commands

endmqtrc	End trace
strmqtrc	Start trace

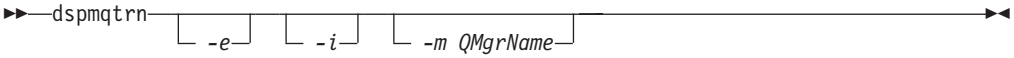
dspmqtrn (Display transactions)

Purpose

Use the **dspmqtrn** command to display details of in-doubt transactions. This includes transactions coordinated by WebSphere MQ and by an external transaction manager.

For each in-doubt transaction, a transaction number (a human-readable transaction identifier), the transaction state, and the transaction ID are displayed. (Transaction IDs can be up to 128 characters long, hence the need for a transaction number.)

Syntax



Optional parameters

-e Requests details of externally coordinated, in-doubt transactions. Such transactions are those for which WebSphere MQ has been asked to prepare to commit, but has not yet been informed of the transaction outcome.

-i Requests details of internally coordinated, in-doubt transactions. Such transactions are those for which each resource manager has been asked to prepare to commit, but WebSphere MQ has yet to inform the resource managers of the transaction outcome.

Information about the state of the transaction in each of its participating resource managers is displayed. This information can help you assess the affects of failure in a particular resource manager.

Note: If you specify neither -e nor -i, details of both internally and externally coordinated in-doubt transactions are displayed.

-m *QMgrName*
The name of the queue manager for which to display transactions. If you omit the name, the default queue manager's transactions are displayed.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

rsvmqtrn	Resolve transaction

endmqcsv (end command server)

Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.

Syntax

Required parameters

QMgrName
The name of the queue manager for which to end the command server.

Optional parameters

-c Stops the command server in a controlled manner. The command server is allowed to complete the processing of any command message that it has already started. No new message is read from the command queue.

This is the default.

-i Stops the command server immediately. Actions associated with a command message currently being processed might not complete.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

1. The following command stops the command server on queue manager `saturn.queue.manager`:

   ```
   endmqcsv  -c saturn.queue.manager
   ```

   The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.
2. The following command stops the command server on queue manager `pluto` immediately:

   ```
   endmqcsv -i pluto
   ```

Related commands

strmqcsv	Start a command server
dspmqcsv	Display the status of a command server

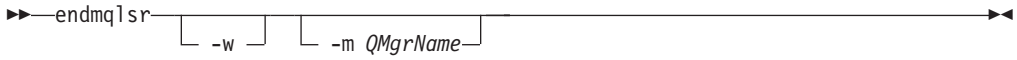
endmqlsr (end listener)

Purpose

The **endmqlsr** command ends all listener processes for the specified queue manager.

Stop the queue manager before issuing the **endmqlsr** command.

Syntax



Optional parameters

-m *QMgrName*
The name of the queue manager. If you omit this, the command operates on the default queue manager.

-w Wait before returning control.

Control is returned to you only after all listeners for the specified queue manager have stopped.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

endmqm (end queue manager)

Purpose

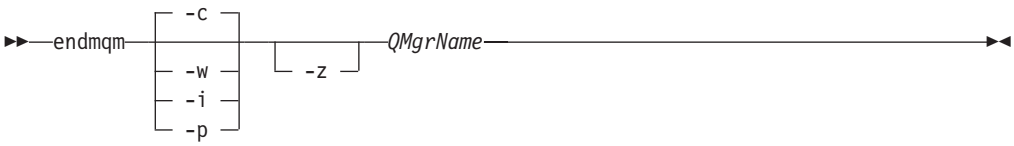
Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Controlled or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Required parameters

QMgrName
The name of the message queue manager to be stopped.

Optional parameters

-c Controlled (or quiesced) shutdown. This is the default.

The queue manager stops, but only after all applications have disconnected. Any MQI calls currently being processed are completed.

Control is returned to you immediately and you are not notified of when the queue manager has stopped.

-w Wait shutdown.

This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped. You receive the message `Waiting for queue manager` *qmName* `to end` while shutdown progresses.

-i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

Control is returned after the queue manager has ended.

-p Preemptive shutdown.

Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager might stop without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for WebSphere MQ applications. The shutdown mode is set to *immediate shutdown*. If the queue manager has not stopped after a few seconds, the shutdown mode is escalated, and all remaining queue manager processes are stopped.

-z Suppresses error messages on the command.

Return codes

0	Queue manager ended
3	Queue manager being created
16	Queue manager does not exist
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error

Examples

The following examples show commands that stop the specified queue managers.

1. This command ends the queue manager named `mercury.queue.manager` in a controlled way. All applications currently connected are allowed to disconnect.

   ```
   endmqm mercury.queue.manager
   ```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

   ```
   endmqm -i saturn.queue.manager
   ```

Related commands

crtmqm	Create a queue manager
strmqm	Start a queue manager
dltmqm	Delete a queue manager

endmqtrc (end trace)

Purpose

The endmqtrc command is not supported by WebSphere MQ for AIX.

Use the **endmqtrc** command to end tracing for the specified entity or all entities.

Syntax

The syntax of this command in WebSphere MQ for HP-UX, Solaris, and Linux is as follows:

The syntax of this command in WebSphere MQ for Windows is as follows:

Optional parameters

The following parameters can be specified in WebSphere MQ for HP-UX, Solaris, and Linux only:

-m *QMgrName*
The name of the queue manager for which to end tracing.

A maximum of one -m flag and associated queue manager name can be supplied on the command.

A queue manager name and -m flag can be specified on the same command as the -e flag.

-e Ends early tracing.

-a Ends all tracing.

This flag ***must*** be specified alone.

Return codes

AMQ5611 This message is issued if you supply invalid arguments to the command.

Examples

This command ends tracing of data for a queue manager called `QM1`.

```
endmqtrc -m QM1
```

Related commands

dspmqtrc	Display formatted trace output
strmqtrc	Start trace

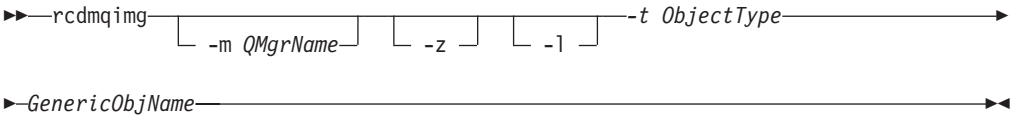
rcdmqimg (record media image)

Purpose

Use the **rcdmqimg** command to write an image of an object, or group of objects, to the log for use in media recovery. Use the associated command **rcrmqobj** to recreate the object from the image.

You use this command with an active queue manager. Further activity on the queue manager is logged so that, although the image becomes out of date, the log records reflect any changes to the object.

Syntax



Required parameters

GenericObjName
The name of the object to record. This parameter can have a trailing asterisk to record that any objects with names matching the portion of the name before the asterisk.

This parameter is required ***unless*** you are recording a queue manager object or the channel synchronization file. Any object name you specify for the channel synchronization file is ignored.

-t *ObjectType*
The types of object for which to record images. Valid object types are:

nl or **namelist**	Namelists
prcs or **process**	Processes
q or **queue**	All types of queue
ql or **qlocal**	Local queues
qa or **qalias**	Alias queues
qr or **qremote**	Remote queues
qm or **qmodel**	Model queues
qmgr	Queue manager object
syncfile	Channel synchronization file
ctlg or **catalog**	An object catalog
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
* or **all**	All the above

Note: When using WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. How you do this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

Optional parameters

-m *QMgrName*
The name of the queue manager for which to record images. If you omit this, the command operates on the default queue manager.

-z Suppresses error messages.

-l Writes messages containing the names of the oldest log files needed to restart the queue manager and to perform media recovery. The messages are written to the error log and the standard error destination. (If you specify both the -z and -l parameters, the messages are sent to the error log, but not to the standard error destination.)

When issuing a sequence of **rcdmqimg** commands, include the -l parameter only on the last command in the sequence, so that the log file information is gathered only once.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
68	Media recovery not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
131	Resource problem
132	Object damaged
135	Temporary object cannot be recorded

Examples

The following command records an image of the queue manager object `saturn.queue.manager` in the log.

```
rcdmqimg -t qmgr -m saturn.queue.manager
```

Related commands

rcrmqobj	Recreate a queue manager object

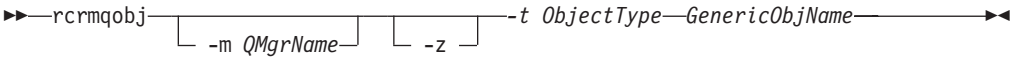
rcrmqobj (recreate object)

Purpose

Use the **rcrmqobj** command to recreate an object, or group of objects, from their images contained in the log. Use the associated command, **rcdmqimg**, to record the object images to the log.

Use this command on a running queue manager. All activity on the queue manager after the image was recorded is logged. To re-create an object, replay the log to re-create events that occurred after the object image was captured.

Syntax



Required parameters

GenericObjName
The name of the object to re-create. This parameter can have a trailing asterisk to re-create any objects with names matching the portion of the name before the asterisk.

This parameter is required ***unless*** the object type is the channel synchronization file; any object name supplied for this object type is ignored.

-t ObjectType
The types of object to re-create. Valid object types are:

nl or **namelist**	Namelists
prcs or **process**	Processes
q or **queue**	All types of queue
ql or **qlocal**	Local queues
qa or **qalias**	Alias queues
qr or **qremote**	Remote queues
qm or **qmodel**	Model queues
syncfile	Channel synchronization file
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
* or **all**	All the above

Note: When using WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. How you do this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

Optional parameters

-m *QMgrName*
The name of the queue manager for which to re-create objects. If omitted, the command operates on the default queue manager.

-z Suppresses error messages.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
66	Media image not available
68	Media recovery not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
135	Temporary object cannot be recovered
136	Object in use

Examples

1. The following command re-creates all local queues for the default queue manager:

   ```
   rcrmqobj -t ql *
   ```

2. The following command re-creates all remote queues associated with queue manager `store`:

   ```
   rcrmqobj -m store -t qr *
   ```

Related commands

rcdmqimg	Record an object in the log

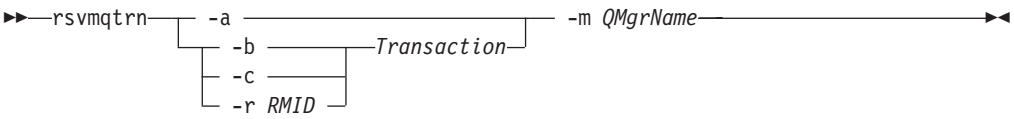
rsvmqtrn (resolve transactions)

Purpose

Use the **rsvmqtrn** command to commit or back out internally or externally coordinated in-doubt transactions.

Use this command only when you are certain that transactions cannot be resolved by the normal protocols. Issuing this command might result in the loss of transactional integrity between resource managers for a distributed transaction.

Syntax



Required parameters

-m *QMgrName*
The name of the queue manager.

Optional parameters

-a The queue manager resolves all internally-coordinated, in-doubt transactions (that is, all global units of work).

-b Backs out the named transaction. This flag is valid for externally-coordinated transactions (that is, for external units of work) only.

-c Commits the named transaction. This flag is valid for externally-coordinated transactions (that is, external units of work) only.

-r *RMID*
The resource manager whose participation in the in-doubt transaction can be ignored. This flag is valid for internally-coordinated transactions only, and for resource managers that have had their resource manager configuration entries removed from the queue manager configuration information.

Note: The queue manager does not call the resource manager. Instead, it marks the resource manager's participation in the transaction as being complete.

Transaction
The transaction number of the transaction being committed or backed out. Use the **dspmqtrn** command to find the relevant transaction number. This parameter is required with the -b, -c, and -r *RMID* parameters.

Return codes

0	Successful operation
32	Transactions could not be resolved
34	Resource manager not recognized
35	Resource manager not permanently unavailable
36	Invalid arguments supplied
40	Queue manager not available

49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

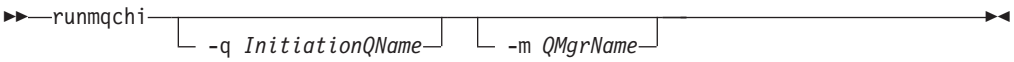
dspmqtrn	Display list of prepared transactions

runmqchi (run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to *WebSphere MQ Intercommunication*.

Syntax



Optional parameters

-q *InitiationQName*
The name of the initiation queue to be processed by this channel initiator. If you omit it, SYSTEM.CHANNEL.INITQ is used.

-m *QMgrName*
The name of the queue manager on which the initiation queue exists. If you omit the name, the default queue manager is used.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

If errors occur that result in return codes of either 10 or 20, review the queue manager error log that the channel is associated with for the error messages, and the @SYSTEM error log for records of problems that occur before the channel is associated with the queue manager. For more information about error logs, see “Error logs” on page 225.

runmqchl (run channel)

Purpose

Use the **runmqchl** command to run either a sender (SDR) or a requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax

Required parameters

-c *ChannelName*
The name of the channel to run.

Optional parameters

-m *QMgrName*
The name of the queue manager with which this channel is associated. If you omit the name, the default queue manager is used.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

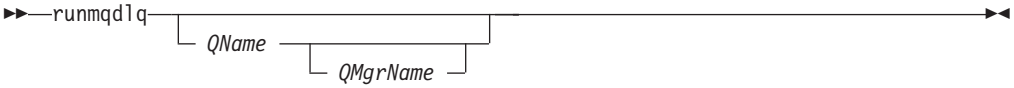
If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages and the @SYSTEM error log for records of problems that occur before the channel is associated with the queue manager.

runmqdlq (run dead-letter queue handler)

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, which monitors and handles messages on a dead-letter queue.

Syntax



Description

Use the dead-letter queue handler to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from `stdin`. When the command is processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter **runmqdlq** rules interactively.

By redirecting the input from a file, you can apply a rules table to the specified queue. The rules table must contain at least one rule.

If you use the DLQ handler without redirecting stdin from a file (the rules table), the DLQ handler reads its input from the keyboard. In WebSphere MQ for AIX, Solaris, HP-UX, and Linux, the DLQ handler does not start to process the named queue until it receives an end_of_file (Ctrl+D) character. In WebSphere MQ for Windows, it does not start to process the named queue until you press the following sequence of keys: Ctrl+Z, Enter, Ctrl+Z, Enter.

For more information about rules tables and how to construct them, see "The DLQ handler rules table" on page 168.

Optional parameters

The MQSC command rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName
The name of the queue to be processed.

If you omit the name, the dead-letter queue defined for the local queue manager is used. If you enter one or more blanks (' '), the dead-letter queue of the local queue manager is explicitly assigned.

QMgrName
The name of the queue manager that owns the queue to be processed.

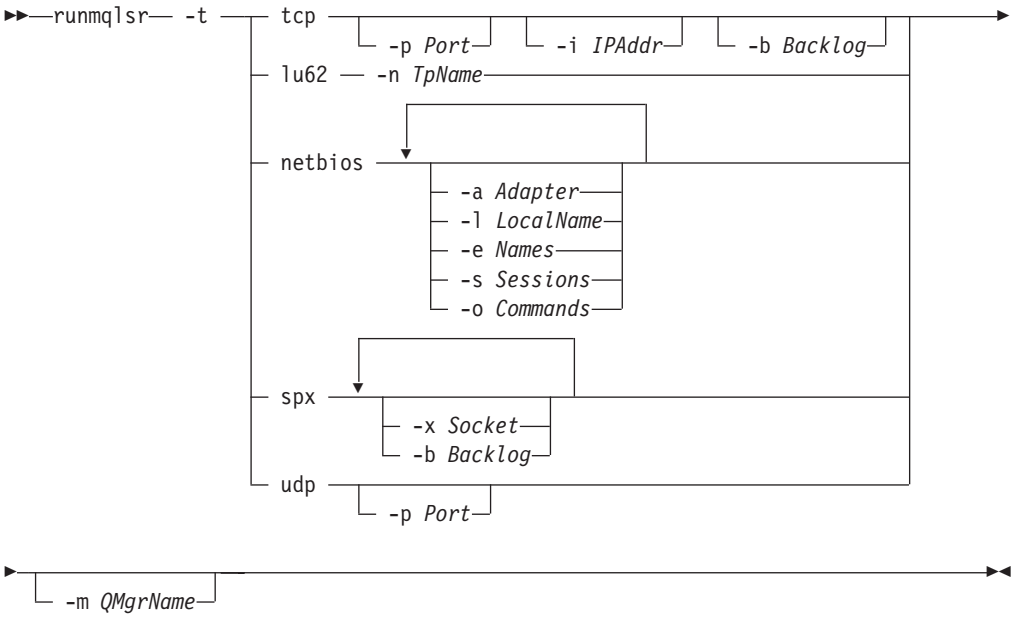
If you omit the name, the default queue manager for the installation is used. If you enter one or more blanks (' '), the default queue manager for this installation is explicitly assigned.

runmqlsr (run listener)

Purpose

Use the **runmqlsr** command to start a listener process.

Syntax



Required parameters

-t The transmission protocol to be used:

`tcp`	Transmission Control Protocol / Internet Protocol (TCP/IP)
`lu62`	SNA LU 6.2 (Windows NT and Windows 2000 only)
`netbios`	NetBIOS (Windows NT and Windows 2000 only)
`spx`	SPX (Windows NT and Windows 2000 only)
`udp`	User datagram protocol (UDP) (AIX only)

Optional parameters

-p *Port*
The port number for TCP/IP. This flag is valid for TCP and UDP. If you omit the port number, it is taken from the queue manager configuration information, or from defaults in the program. The default value is 1414.

-i *IPAddr*
The IP address for the listener, specified in dotted decimal or alphanumeric format. This flag is valid only for TCP/IP. If you omit this parameter, the listener listens on all IP addresses available to the TCP/IP stack.

-n *TpName*
The LU 6.2 transaction program name. This flag is valid only for the LU 6.2 transmission protocol. If you omit the name, it is taken from the queue manager configuration information.

-a *Adapter*
The adapter number on which NetBIOS listens. By default the listener uses adapter 0.

-l *LocalName*
The NetBIOS local name that the listener uses. The default is specified in the queue manager configuration information.

-e *Names*
The number of names that the listener can use. The default value is specified in the queue manager configuration information.

-s *Sessions*
The number of sessions that the listener can use. The default value is specified in the queue manager configuration information.

-o *Commands*
The number of commands that the listener can use. The default value is specified in the queue manager configuration information.

-x *Socket*
The SPX socket on which SPX listens. The default value is hexadecimal 5E86.

-m *QMgrName*
The name of the queue manager. By default the command operates on the default queue manager.

-b *Backlog*
The number of concurrent connection requests that the listener supports. See "LU62, NETBIOS, TCP, and SPX" on page 107 for a list of default values and further information.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. The listener can use a maximum of five names, five commands, and five sessions. These resources must be within the limits set in the queue manager configuration information.

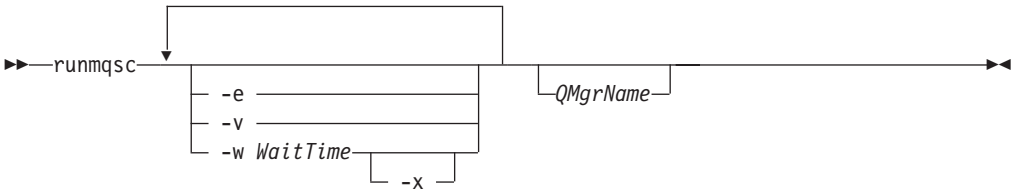
```
runmqlsr -t netbios -e 5 -s 5 -o 5
```

runmqsc (run MQSC commands)

Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks, for example defining, altering, or deleting a local queue object. MQSC commands and their syntax are described in the *WebSphere MQ Script (MQSC) Command Reference*.

Syntax



Description

You can invoke the **runmqsc** command in three ways:

Verify command
Verify MQSC commands but do not run them. An output report is generated indicating the success or failure of each command. This mode is available on a local queue manager only.

Run command directly
Send MQSC commands directly to a local queue manager.

Run command indirectly
Run MQSC commands on a remote queue manager. These commands are put on the command queue on a remote queue manager and run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

Indirect mode operation is performed through the default queue manager.

The **runmqsc** command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file, you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

Optional parameters

-e Prevents source text for the MQSC commands from being copied into a report. This is useful when you enter commands interactively.

-v Verifies the specified commands without performing the actions. This mode is only available locally. The -w and -x flags are ignored if they are specified at the same time.

-w *WaitTime*
Run the MQSC commands on another queue manager. You must have the

required channel and transmission queues set up for this. See "Preparing channels and transmission queues for remote administration" on page 62 for more information.

WaitTime
The time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, but the MQSC commands still run. Specify a time between 1 and 999 999 seconds.

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

Indirect mode operation is performed through the default queue manager.

This flag is ignored if the -v flag is specified.

-x The target queue manager is running under z/OS. This flag applies only in indirect mode. The -w flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the WebSphere MQ for z/OS command queue.

QMgrName
The name of the target queue manager on which to run the MQSC commands, by default, the default queue manager.

Return codes

00	MQSC command file processed successfully
10	MQSC command file processed with errors; report contains reasons for failing commands
20	Error; MQSC command file not run

Examples

1. Enter this command at the command prompt:

   ```
   runmqsc
   ```

   Now you can enter MQSC commands directly at the command prompt. No queue manager name is specified, so the MQSC commands are processed on the default queue manager.
2. Use one of these commands, as appropriate in your environment, to specify that MQSC commands are to be verified only:

   ```
   runmqsc -v BANK < "/u/users/commfile.in"

   runmqsc -v BANK < "c:\users\commfile.in"
   ```

   This command verifies the MQSC commands in file `commfile.in`. The queue manager name is `BANK`. The output is displayed in the current window.
3. These commands run the MQSC command file `mqscfile.in` against the default queue manager.

   ```
   runmqsc < "/var/mqm/mqsc/mqscfile.in" > "/var/mqm/mqsc/mqscfile.out"

   runmqsc < "c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.in" >
    "c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.out"
   ```

In this example, the output is directed to file `mqscfile.out`.

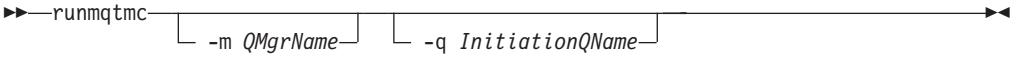
runmqtmc (start client trigger monitor)

Purpose

The runmqtmc command is available on AIX clients only.

Use the **runmqtmc** command to invoke a trigger monitor for a client. For further information about using trigger monitors, refer to the *WebSphere MQ Application Programming Guide*.

Syntax



Optional parameters

-m *QMgrName*
The name of the queue manager on which the client trigger monitor operates, by default the default queue manager.

-q *InitiationQName*
The name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

Return codes

0	Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.
10	Client trigger monitor interrupted by an error.
20	Error; client trigger monitor not run.

Examples

For examples of using this command, refer to the *WebSphere MQ Application Programming Guide*.

runmqtrm (start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the *WebSphere MQ Application Programming Guide*.

Syntax

Optional parameters

-m *QMgrName*
The name of the queue manager on which the trigger monitor operates, by default the default queue manager.

-q *InitiationQName*
Specifies the name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

Return codes

0	Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
10	Trigger monitor interrupted by an error.
20	Error; trigger monitor not run.

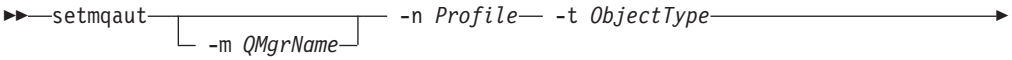
setmqaut (set or reset authority)

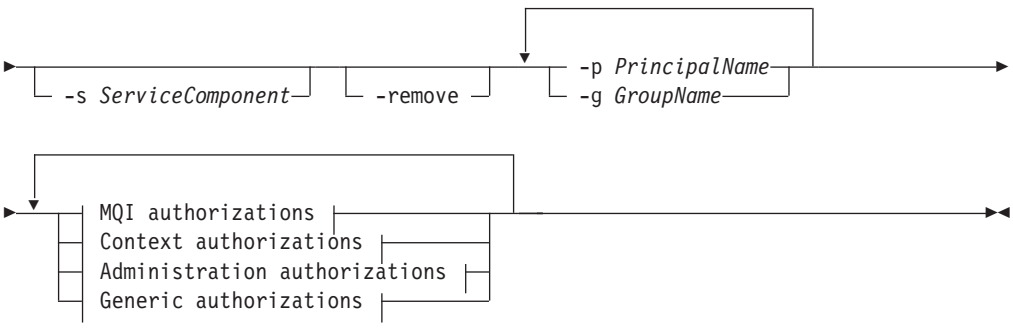
Purpose

Use the **setmqaut** command to change the authorizations to a profile, object or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

For more information about authorization service components, see "Installable services" on page 100, "Service components" on page 101, and Chapter 20, "Authorization service" on page 341.

Syntax





MQI authorizations:

Context authorizations:

Administration authorizations:

Generic authorizations:

Description

Use setmqaut both to *set* an authorization, that is, give a user group or principal permission to perform an operation, and to *reset* an authorization, that is, remove the permission to perform an operation. You must specify the user groups and principals to which the authorizations apply, the queue manager, object type, and the profile name identifying the object or objects. You can specify any number of groups and principals in a single command.

Note: In WebSphere MQ for UNIX systems, if you specify a set of authorizations for a principal, the same authorizations are given to all principals in the same primary group.

The authorizations that can be given are categorized as follows:
- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by + or −. For example, if you include `+put` in the authorization list, you give authority to issue **MQPUT** calls against a queue. Alternatively, if you include `-put` in the authorization list, you remove the authorization to issue **MQPUT** calls.

Authorizations can be specified in any order provided that they do not clash. For example, specifying `allmqi` with `set` causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

Required parameters

-t *ObjectType*
The type of object for which to change authorizations.

Possible values are:
- **q** or **queue**
- **prcs** or **process**
- **qmgr**
- **nl** or **namelist**
- **authinfo** (for use with Secure Sockets Layer (SSL) channel security)

-n *Profile*
The name of the profile for which to change authorizations. The authorizations apply to all WebSphere MQ objects with names that match the profile name specified. The profile name can be generic, using wildcard characters to specify a range of names as explained in "Using OAM generic profiles" on page 123.

If you give an explicit profile name (without any wildcard characters), the object identified must exist.

This parameter is required, unless you are changing the authorizations of your default queue manager, in which case you ***must not*** include it.

Optional parameters

-m *QMgrName*
The name of the queue manager of the object for which to change authorizations. The name can contain up to 48 characters.

This parameter is optional if you are changing the authorizations of your default queue manager.

-p *PrincipalName*
The name of the principal for which to change authorizations.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see "Principals and groups" on page 116.

You must have at least one principal or group.

-g *GroupName*
The name of the user group for which to change authorizations. You can specify more than one group name, but each name must be prefixed by the -g flag. On Windows systems, you can use only local groups.

-s *ServiceComponent*
The name of the authorization service to which the authorizations apply (if your system supports installable authorization services). This parameter is optional; if you omit it, the authorization update is made to the first installable component for the service.

-remove
Removes a profile. The authorizations associated with the profile no longer apply to WebSphere MQ objects with names that match the profile name specified.

Authorizations
The authorizations to be given or removed. Each item in the list is prefixed by a + indicating that authority is to be given, or a −, indicating that authority is to be removed.

For example, to give authority to issue an **MQPUT** call from the MQI, specify `+put` in the list. To remove authority to issue an **MQPUT** call, specify `-put`.

Table 22 shows the authorities that can be given to the different object types.

Table 22. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Namelist	Authentication information
all	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes
none	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No
browse	Yes	No	No	No	No
chg	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No
connect	No	No	Yes	No	No
crt	Yes	Yes	Yes	Yes	Yes
dlt	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No
put	Yes	No	No	No	No
inq	Yes	Yes	Yes	Yes	Yes
passall	Yes	No	No	No	No
passid	Yes	No	No	No	No
set	Yes	No	No	No	No
setall	Yes	No	No	No	No
setid	Yes	No	No	No	No

Authorizations for MQI calls

altusr	Use another user's authority for **MQOPEN** and **MQPUT1** calls.
browse	Retrieve a message from a queue using an **MQGET** call with the BROWSE option.
connect	Connect the application to the specified queue manager using an **MQCONN** call.
get	Retrieve a message from a queue using an **MQGET** call.
inq	Make an inquiry on a specific queue using an **MQINQ** call.
put	Put a message on a specific queue using an **MQPUT** call.
set	Set attributes on a queue from the MQI using an **MQSET** call.

Note: If you open a queue for multiple options, you have to be authorized for each option.

Authorizations for context

passall	Pass all context on the specified queue. All the context fields are copied from the original request.
passid	Pass identity context on the specified queue. The identity context is the same as that of the request.
setall	Set all context on the specified queue. This is used by special system utilities.
setid	Set identity context on the specified queue. This is used by special system utilities.

Authorizations for commands

chg	Change the attributes of the specified object.
clr	Clear the specified queue (PCF Clear queue command only).
crt	Create objects of the specified type.
dlt	Delete the specified object.
dsp	Display the attributes of the specified object.

Authorizations for generic operations

all	Use all operations applicable to the object.
alladm	Use all administration operations applicable to the object.
allmqi	Use all MQI calls applicable to the object.
none	No authority. Use this to create profiles without authority.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type

149	Entity name missing
150	Authorization specification missing
151	Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue `orange.queue` on queue manager `saturn.queue.manager`. If the queue does not exist, the command fails.

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g tango +inq +alladm
   ```

   The authorizations are given to user group `tango` and the associated authorization list specifies that user group `tango` can:
   - Issue **MQINQ** calls
   - Perform all administration operations on that object
2. In this example, the authorization list specifies that user group `foxy`:
   - Cannot issue any calls from the MQI to the specified queue
   - Can perform all administration operations on the specified queue

   If the queue does not exist, the command fails.

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g foxy -allmqi +alladm
   ```
3. This example gives `user1` full access to all queues with names beginning `a.b` on queue manager `qmgr1`. The profile is persistent, and will apply to any object with a name that matches the profile name.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +all
   ```
4. This example deletes the specified profile.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 -remove
   ```
5. This example creates a profile with no authority.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +none
   ```

Related commands

dmpmqaut	Dump authority
dspmqaut	Display authority

setmqcap (set license units)

Purpose

Use the **setmqcap** command to set your current purchased license units, by specifying the number of processors on which you are running WebSphere MQ.

To fulfil the conditions of your license agreement with IBM, you must have purchased sufficient license units for the number of processors on which you are running WebSphere MQ. The installation dialog checks for this when you install WebSphere MQ, and you can set or reset it using the **setmqcap** command.

Syntax

Required parameters

CapUnits
A positive integer, stating the number of processors for which you have purchased license units. If your license units are not sufficient for the number of online processors on which you are running WebSphere MQ, a warning message (`insufficient license units`) is output. A message is placed in the error log indicating the value entered on the command.

Use a value of -1 to specify that you have *group enablement*, which allows use of the server through an enterprise-wide agreement, or other multi-server agreement, not administered or tracked at the individual server level. This value disables further checking of capacity units on this machine.

The default value, zero, indicates that you have purchased the media with no license units.

Return codes

0	Successful operation
36	Invalid argument
71	Unexpected error

Related commands

dspmqcap	Display license units

setmqcrl (set certificate revocation list (CRL) LDAP server definitions)

Purpose

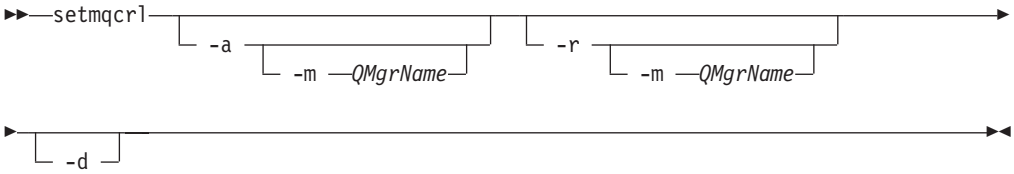
The setmqcrl command applies to WebSphere MQ for Windows only.

Use the **setmqcrl** command to configure and administer support for publishing CRL (certificate revocation list) LDAP definitions in an Active Directory.

A domain administrator must use this command, or **setmqscp**, initially to prepare the Active Directory for WebSphere MQ usage and to grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects. You can also use the **setmqcrl** command to display all the currently configured CRL server definitions available on the Active Directory, that is, those definitions referred to by the queue manager's CRL namelist.

The only types of CRL servers supported are LDAP servers.

Syntax



Optional parameters

You must specify one of `-a` (add), `-r` (remove) or `-d` (display).

-a Adds the WebSphere MQ client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the **setmqscp** command.

-d Displays the WebSphere MQ CRL server definitions.

-r Removes the WebSphere MQ CRL server definitions.

-m [* | qmgr]
Modifies the specified parameter (`-a` or `-r`) so that only the specified queue manager is affected. You must include this option with the `-a` parameter.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific WebSphere MQ CRL server definitions file from one queue manager alone.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently. (In this, it is functionally equivalent to `setmqscp -a`.)

```
setmqcrl -a
```

setmqcrl

The following command migrates existing CRL server definitions from a local queue manager, Paint.queue.manager, to the Active Directory, **deleting any other CRL definitions from the Active Directory first**:

```
setmqcrl -a -m Paint.queue.manager
```

setmqscp (set service connection points)

Purpose

The setmqscp command applies to WebSphere MQ for Windows only.

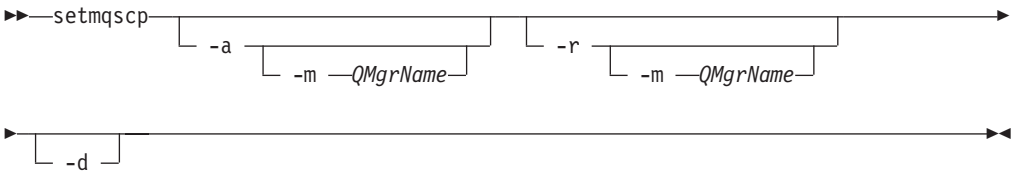
Use the **setmqscp** command to configure and administer support for publishing client connection channel definitions in an Active Directory.

Initially, this command is used by a domain administrator to:

- Prepare the Active Directory for WebSphere MQ use
- Grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects

You can also use the **setmqscp** command to display all the currently configured client connection channel definitions available on the Active Directory.

Syntax



Optional parameters

You must specify one of -a (add), -r (remove) or -d (display).

-a Adds the WebSphere MQ client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the `setmqscp -r` command.

-d Displays the service connection points.

-r Removes the service connection points. If you omit -m, and no client connection definitions exist in the IBM-MQClientConnections folder, the folder itself is removed from the Active Directory.

-m [* | qmgr]
Modifies the specified parameter (-a or -r) so that only the specified queue manager is affected.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific client connection table file from one queue manager alone, if required.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently:

```
setmqscp -a
```

setmqscp

The following command migrates existing client connection definitions from a local queue manager, Paint.queue.manager, to the Active Directory:

```
setmqscp -a -m Paint.queue.manager
```

The following command migrates all client connection definitions on the local server to the Active Directory:

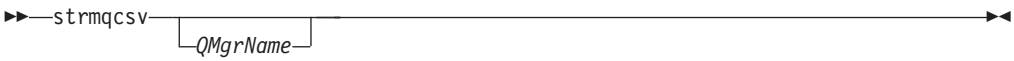
```
setmqscp -a -m *
```

strmqcsv (start command server)

Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables WebSphere MQ to process commands sent to the command queue.

Syntax



Required parameters

None

Optional parameters

QMgrName
The name of the queue manager for which to start the command server.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command starts a command server for queue manager `earth`:

```
strmqcsv earth
```

Related commands

endmqcsv	End a command server
dspmqcsv	Display the status of a command server

strmqm (start queue manager)

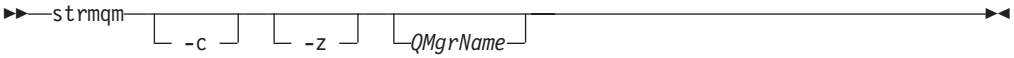
Purpose

Use the **strmqm** command to start a local queue manager.

When you run **strmqm**, WebSphere MQ checks the setting for license units. If the purchased processor capacity is not -1, it compares the number it finds with the number of processors. If the number is insufficient it issues an appropriate warning message, as follows:

- `Insufficient license units` (if you have run **setmqcap**)
- `Purchased processor capacity not set (by using setmqcap)` (if you have not run **setmqcap**)

Syntax



Optional parameters

-c Starts the queue manager, redefines the default and system objects, then stops the queue manager. (Use the **crtmqm** command to create the default and system objects for a queue manager.) Any existing system and default objects belonging to the queue manager are replaced if you specify this flag.

-z Suppresses error messages.

This flag is used within WebSphere MQ to suppress unwanted error messages. Because using this flag could result in loss of information, do not use it when entering commands on a command line.

QMgrName
The name of a local queue manager to start, by default the default queue manager.

Return codes

0	Queue manager started
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
23	Log not available
24	A process that was using the previous instance of the queue manager has not yet disconnected.
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid

Examples

The following command starts the queue manager `account`:

```
strmqm account
```

Related commands

crtmqm	Create a queue manager
dltmqm	Delete a queue manager
endmqm	End a queue manager

strmqtrc (Start trace)

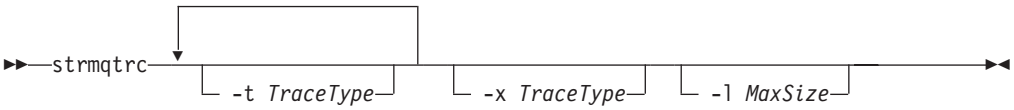
Purpose

The strmqtrc command is not supported by WebSphere MQ for AIX. (See "Tracing WebSphere MQ for AIX" on page 229.)

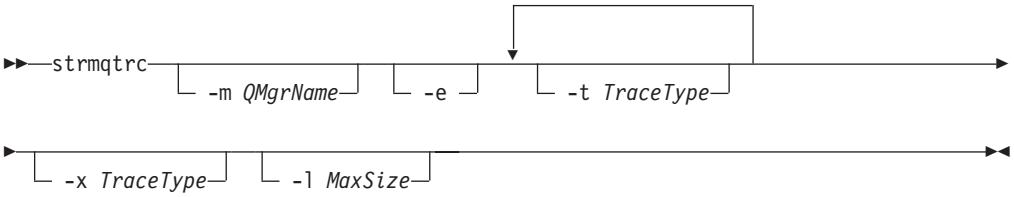
Use the **strmqtrc** command to enable tracing. This command can be run regardless of whether tracing is enabled. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

Syntax

The syntax of this command in WebSphere MQ for Windows is as follows:



The syntax of this command in WebSphere MQ for HP-UX, Solaris, and Linux is as follows:



Description

You can request different levels of trace detail. For each *tracetype* value you specify, including `-t all`, specify either `-t parms` or `-t detail` to obtain the appropriate level of trace detail. If you do not specify either `-t parms` or `-t detail` for any particular trace type, only a default-detail trace is generated for that trace type.

You can use the `-x` flag with *tracetype* values to exclude those entry points you do not want to record. This is useful in reducing the amount of trace produced.

In WebSphere MQ for Windows, the output file is created in the `\<mqmwork>\errors` directory, where `<mqmwork>` is the directory selected to hold WebSphere MQ data files.

In WebSphere MQ for HP-UX, Solaris, and Linux, the output file is always created in the directory `/var/mqm/trace`.

For examples of trace data generated by this command see "Tracing" on page 228.

Optional parameters

-m *QMgrName*
The name of the queue manager to trace.

A queue manager name and the -m flag can be specified on the same command as the -e flag. If more than one trace specification applies to a given entity being traced, the actual trace includes all the specified options.

It is an error to omit the -m flag and queue manager name, unless you specify the -e flag.

This parameter is not valid in WebSphere MQ for Windows.

-e Requests early tracing, making it possible to trace the creation or startup of a queue manager. If you include this flag, any process belonging to any component of any queue manager traces its early processing. The default is not to perform early tracing.

This parameter is not valid in WebSphere MQ for Windows.

-t *TraceType*

The points to trace and the amount of trace detail to record. By default **all** trace points are enabled and a default-detail trace is generated.

Alternatively, you can supply one or more of the options in the following list.

If you supply multiple trace types, each must have its own -t flag. You can include any number of -t flags, provided that each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

all	Output data for every trace point in the system (the default). The all parameter activates tracing at default detail level.
api	Output data for trace points associated with the MQI and major queue manager components.
commentary	Output data for trace points associated with comments in the WebSphere MQ components.
comms	Output data for trace points associated with data flowing over communications networks.
csdata	Output data for trace points associated with internal data buffers in common services.
csflows	Output data for trace points associated with processing flow in common services.
detail	Activate tracing at high-detail level for flow processing trace points.
lqmdata	Output data for trace points associated with internal data buffers in the local queue manager.
lqmflows	Output data for trace points associated with processing flow in the local queue manager.
otherdata	Output data for trace points associated with internal data buffers in other components.
otherflows	Output data for trace points associated with processing flow in other components.
parms	Activate tracing at default-detail level for flow processing trace points.
remotedata	Output data for trace points associated with internal data buffers in the communications component.
remoteflows	Output data for trace points associated with processing flow in the communications component.

servicedata Output data for trace points associated with internal data buffers in the service component.

serviceflows Output data for trace points associated with processing flow in the service component.

ssl Output data associated with using GSKit to enable Secure Sockets Layer (SSL) channel security.

This parameter is not valid in WebSphere MQ for Windows.

versiondata Output data for trace points associated with the version of WebSphere MQ running.

-x *TraceType*
The points **not** to trace. By default **all** trace points are enabled and a default-detail trace is generated. The trace points you can specify are those listed for the -t flag.

If you supply multiple trace types, each must have its own -x flag. You can include any number of -x flags, provided that each has a valid trace type associated with it.

-l *MaxSize*
The maximum size of a trace file (AMQnnnn.TRC) in millions of bytes. For example, if you specify a MaxSize of 1, the size of the trace is limited to 1 MB.

When a trace file reaches the specified maximum, it is renamed to AMQnnnn.TRS and a new AMQnnnn.TRC file is started. All trace files are restarted when the maximum limit is reached. If a previous copy of an AMQnnnn.TRS file exists, it is deleted.

Return codes

AMQ7024	Non-valid arguments supplied to the command.
AMQ8304	Nine concurrent traces (the maximum) already running.

Examples

This command enables tracing of processing flow from common services and the local queue manager for a queue manager called QM1 in WebSphere MQ for HP-UX, Solaris, and Linux. Trace data is generated at the default level of detail.

```
strmqtrc -m QM1 -t csflows -t lqmflows -t parms
```

This command disables tracing of SSL activity on a queue manager called QM1 in WebSphere MQ for HP-UX, Solaris, and Linux. Other trace data is generated at the parms level of detail.

```
strmqtrc -m QM1 -x ssl -t parms
```

This command enables high-detail tracing of the processing flow for all components in WebSphere MQ for Windows:

```
strmqtrc -t all -t detail
```

Related commands

dspmqtrc	Display formatted trace output
endmqtrc	End trace

Chapter 18. Using the IKEYCMD interface to manage keys and certificates on UNIX systems

IKEYCMD is a UNIX command line interface that you can use to manage keys, certificates, and certificate requests. IKEYCMD provides functions that are similar to those of the iKeyman GUI, described in *WebSphere MQ Security*. You can call IKEYCMD from a shell script when you want to create your own interface for certificate and key management tasks. Use IKEYCMD to:

- Create the type of CMS key database files that WebSphere MQ requires
- Create certificate requests
- Import personal certificates
- Import CA certificates
- Manage self-signed certificates

This section tells you about:

Setting up to use IKEYCMD

To run the IKEYCMD command line interface, set up environment variables as follows:

1. Set the JAVA_HOME environment variable:

AIX	`export JAVA_HOME=/usr/mqm/ssl/jre`
HP-UX	`export JAVA_HOME=/opt/mqm/ssl`
Linux	`export JAVA_HOME=/opt/mqm/ssl/jre`
Solaris	`export JAVA_HOME=/opt/mqm/ssl`

2. Set your PATH to include your JRE executables:

```
export PATH=$JAVA_HOME/bin:$PATH
```

3. Set the CLASSPATH environment variable:

AIX	`export CLASSPATH=/usr/opt/ibm/gskak/classes/cfwk.zip:$CLASSPATH` `export CLASSPATH=/usr/opt/ibm/gskak/classes/gsk6cls.jar:$CLASSPATH`
HP-UX	`export CLASSPATH=/opt/ibm/gsk6/classes/cfwk.zip:$CLASSPATH` `export CLASSPATH=/opt/ibm/gsk6/classes/gsk6cls.jar:$CLASSPATH`
Linux	`export CLASSPATH=/usr/local/ibm/gsk6/classes/cfwk.zip:$CLASSPATH` `export CLASSPATH=/usr/local/ibm/gsk6/classes/gsk6cls.jar:$CLASSPATH`
Solaris	`export CLASSPATH=/opt/ibm/gsk6/classes/cfwk.zip:$CLASSPATH` `export CLASSPATH=/opt/ibm/gsk6/classes/gsk6cls.jar:$CLASSPATH`

4. **If you are using Linux for zSeries**, set the LD_PRELOAD environment variable:

```
export LD_PRELOAD=/usr/lib/libstdc++-libc6.1-2.so.3
```

IKEYCMD syntax

Use the gsk6cmd script to perform tasks with the IKEYCMD interface.

Note: The full syntax of the IKEYCMD command line interface is:

```
java [-Dikeycmd.properties=properties_file] com.ibm.gsk.ikeyman.ikeycmd
     object action [options]
```

where `-Dikeycmd.properties` specifies the name of an optional properties file to use for this invocation of IKEYCMD. A default properties file, `ikeycmd.properties`, is provided as a sample file that you can modify.

Each command specifies at least one *object*. Commands for PKCS #11 device operations might specify additional objects. Commands for key database, certificate, and certificate request objects also specify an *action*.

IKEYCMD commands

This section describes commands according to the object of the command. The object can be one of the following:

–keydb	Actions apply to a key database
–cert	Actions apply to a certificate
–certreq	Actions apply to a certificate request
–help	Displays help for IKEYCMD
–version	Displays version information for IKEYCMD

The following sections describe the actions that you can take on key database, certificate, and certificate request objects:

- "Commands for a CMS key database only"
- "Commands for CMS or PKCS #12 key databases" on page 321
- "Commands for cryptographic device operations" on page 323

See "IKEYCMD options" on page 324 for a description of the options on these commands.

Commands for a CMS key database only

–keydb –changepw

Change the password for a CMS key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password
    -stash -expire days
```

–keydb –create

Create a CMS key database:

```
-keydb -create -db filename -pw password -type cms -expire days -stash
```

–keydb –stashpw

Stash the password of a CMS key database into a file:

```
-keydb -stashpw -db filename -pw password
```

–cert –getdefault

Get the default personal certificate:

```
-cert -getdefault -db filename -pw password
```

–cert –modify
Modify a certificate:

Note: Currently, the only field that can be modified is the Certificate Trust field.

```
-cert -modify -db filename -pw password -label label
    -trust enable | disable
```

–cert –setdefault
Set the default personal certificate:

```
-cert -setdefault -db filename -pw password -label label
```

Commands for CMS or PKCS #12 key databases

–keydb –changepw
Change the password for a key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password -expire days
```

–keydb –convert
Convert the key database from one format to another:

```
-keydb -convert -db filename -pw password
    -old_format cms | pkcs12 -new_format cms
```

–keydb –create
Create a key database:

```
-keydb -create -db filename -pw password -type cms | pkcs12
```

–keydb –delete
Delete a key database:

```
-keydb -delete -db filename -pw password
```

–keydb –list
List currently-supported types of key database:

```
-keydb -list
```

–cert –add
Add a certificate from a file into a key database:

```
-cert -add -db filename -pw password -label label -file filename
    -format ascii | binary
```

–cert –create
Create a self-signed certificate:

```
-cert -create -db filename -pw password -label label -dn distinguished_name
    -size 1024 | 512 -x509version 3 | 1 | 2 -expire days
```

–cert –delete
Delete a certificate:

```
-cert -delete -db filename -pw password -label label
```

–cert –details
List the detailed information for a specific certificate:

```
-cert -details -db filename -pw password -label label
```

–cert –export
Export a personal certificate and its associated private key from a key database into a PKCS#12 file, or to another key database:

```
-cert -export -db filename -pw password -label label -type cms | pkcs12
    -target filename -target_pw password -target_type cms | pkcs12
```

IKEYCMD commands

–cert –extract

Extract a certificate from a key database:

```
-cert -extract -db filename -pw password -label label -target filename
    -format ascii | binary
```

–cert –import

Import a personal certificate from a key database:

```
 -cert -import -file filename -pw password -type pkcs12 -target filename
       -target_pw password -target_type cms
```

–cert –list

List all certificates in a key database:

```
-cert -list all | personal | CA
    -db filename -pw password
```

–cert –receive

Receive a certificate from a file:

```
-cert -receive -file filename -db filename -pw password
    -format ascii | binary -default_cert yes | no
```

–cert –sign

Sign a certificate:

```
-cert -sign -file filename -db filename -pw password
 -label label -target filename
    -format ascii | binary -expire days
```

–certreq –create

Create a certificate request:

```
-certreq -create -db filename -pw password
    -label label -dn distinguished_name
    -size 1024 | 512 -file filename
```

–certreq –delete

Delete a certificate request:

```
-certreq -delete -db filename -pw password -label label
```

–certreq –details

List the detailed information of a specific certificate request:

```
-certreq -details -db filename -pw password -label label
```

List the detailed information about a certificate request and show the full certificate request:

```
-certreq -details -showOID -db filename
    -pw password -label label
```

–certreq –extract

Extract a certificate request from a certificate request database into a file:

```
-certreq -extract -db filename -pw password
     -label label -target filename
```

–certreq –list

List all certificate requests in the certificate request database:

```
-certreq -list -db filename -pw password
```

–certreq –recreate

Recreate a certificate request:

```
-certreq -recreate -dn distinguished_name -pw password
    -label label -target filename
```

Commands for cryptographic device operations

–keydb –changepw

Change the password for a cryptographic device:

```
-keydb -changepw -crypto module_name -tokenlabel token_label
    -pw password -new_pw new_password
```

–keydb –list

List currently-supported types of key database:

```
-keydb -list
```

–cert –add

Add a certificate from a file to a cryptographic device:

```
-cert -add -crypto module_name -tokenlabel token_label
    -pw password -label label -file filename -format ascii | binary
```

–cert –create

Create a self-signed certificate on a cryptographic device:

```
-cert -create -crypto module_name -tokenlabel token_label
    -pw password -label label -dn distinguished_name -size 1024 | 512
    -x509version 3 | 1 | 2 -default_cert no | yes -expire days
```

–cert –delete

Delete a certificate on a cryptographic device:

```
-cert -delete -crypto module_name -tokenlabel token_label
    -pw password -label label
```

–cert –details

List the detailed information for a specific certificate on a cryptographic device:

```
-cert -details -crypto module_name -tokenlabel token_label
    -pw password -label label
```

List the detailed information and show the full certificate for a specific certificate on a cryptographic device:

```
-cert -details -showOID -crypto module_name -tokenlabel token_label
    -pw password -label label
```

–cert –extract

Extract a certificate from a key database:

```
-cert -extract -crypto module_name -tokenlabel token_label
    -pw password -label label -target filename -format ascii | binary
```

–cert –import

Import a certificate to a cryptographic device with secondary key database support:

```
-cert -import -db filename -pw password -label label -type cms
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password
```

Import a PKCS #12 certificate to a cryptographic device with secondary key database support:

```
-cert -import -file filename -pw password -type pkcs12
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password
```

Note: You cannot import a certificate containing multiple OU (organizational unit) attributes in the distinguished name.

IKEYCMD commands

–cert –list

List all certificates on a cryptographic device:

```
-cert -list all | personal | CA
    -crypto module_name -tokenlabel token_label -pw password
```

–cert –receive

Receive a certificate from a file to a cryptographic device with secondary key database support:

```
-cert -receive -file filename -crypto module_name -tokenlabel token_label
    -pw password  -default_cert yes | no
    -secondaryDB filename -secondaryDBpw password -format ascii | binary
```

–certreq –create

Create a certificate request on a cryptographic device:

```
-certreq -create -crypto module_name -tokenlabel token_label
    -pw password -label label -dn distinguished_name
    -size 1024 | 512 -file filename
```

–certreq –delete

Delete a certificate request from a cryptographic device:

```
-certreq -delete -crypto module_name -tokenlabel token_label
    -pw password -label label
```

–certreq –details

List the detailed information of a specific certificate request on a cryptographic device:

```
-certreq -details -crypto module_name -tokenlabel token_label
    -pw password -label label
```

List the detailed information about a certificate request and show the full certificate request on a cryptographic device:

```
-certreq -details -showOID -crypto module_name -tokenlabel token_label
    -pw password -label label
```

–certreq –extract

Extract a certificate request from a certificate request database on a cryptographic device into a file:

```
-certreq -extract -crypto module_name -tokenlabel token_label
    -pw password -label label -target filename
```

–certreq –list

List all certificate requests in the certificate request database on a cryptographic device:

```
-certreq -list -crypto module_name -tokenlabel token_label
    -pw password
```

IKEYCMD options

Table 23 lists the options that can be present on the command line. Note that the meaning of an option can depend on the object and action specified in the command.

Table 23. Options that can be used with the IKEYCMD interface

IKEYCMD option	Description
-crypto	Name of the module to manage a PKCS #11 cryptographic device. The value after **–crypto** is optional if you specify the module name in the properties file

Table 23. Options that can be used with the IKEYCMD interface (continued)

IKEYCMD option	Description
-db	Fully qualified path name of a key database.
-default_cert	Sets a certificate as the default certificate. The value can be `yes` or `no`. The default is `no`.
-dn	X.500 distinguished name. The value is a string enclosed in double quotes, for example "`CN=John Smith,O=IBM ,OU=Test ,C=GB`". Note that only the CN, O, and C attributes are required. **Note:** Avoid using multiple OU attributes in distinguished names when you create self-signed certificates. When you create such certificates, only the **last entered** OU value is accepted into the certificate.
-encryption	Strength of encryption used in certificate export command. The value can be `strong` or `weak`. The default is `strong`.
-expire	Expiration time in days of either a certificate or a database password. The defaults are 365 days for a certificate and 60 days for a database password.
-file	File name of a certificate or certificate request.
-format	Format of a certificate. The value can be `ascii` for Base64_encoded ASCII or `binary` for Binary DER data. The default is `ascii`.
-label	Label attached to a certificate or certificate request.
-new_format	New format of key database.
-new_pw	New database password.
-old_format	Old format of key database.
-pw	Password for the key database or PKCS#12 file.
-secondaryDB	Name of a secondary key database for PKCS #11 device operations.
-secondaryDBpw	Password for the secondary key database for PKCS #11 device operations.
-showOID	Displays the full certificate or certificate request.
-size	Key size. The value can be 512 or 1024. The default is 1024.
-stash	Tells IKEYCMD to stash the key database password to a file.
-target	Destination file or database.
-target_pw	Password for the key database if -target specifies a key database.
-target_type	Type of database specified by -target operand. See -type option for permitted values.
-tokenLabel	Label of a PKCS #11 cryptographic device.
-trust	Trust status of a CA certificate. The value can be `enable` or `disable`. The default is `enable`.
-type	Type of database. The value can be: • `cms` for a CMS key database • `pkcs12` for a PKCS#12 file
-x509version	Version of X.509 certificate to create. The value can be 1, 2, or 3. The default is 3.

IKEYCMD options

Part 7. WebSphere MQ installable services and the API exit

Chapter 19. Installable services and components

This chapter introduces the installable services and the functions and components associated with them. We document the interface to these functions so that you, or software vendors, can supply components.

The chapter includes:

The installable services interface is described in Chapter 22, "Installable services interface reference information" on page 353.

Why installable services?

The main reasons for providing WebSphere MQ installable services are:

- To provide you with the flexibility of choosing whether to use components provided by WebSphere MQ products, or replace or augment them with others.
- To allow vendors to participate, by providing components that might use new technologies, without making internal changes to WebSphere MQ products.
- To allow WebSphere MQ to exploit new technologies faster and cheaper, and so provide products earlier and at lower prices.

Installable services and *service components* are part of the WebSphere MQ product structure. At the center of this structure is the part of the queue manager that implements the function and rules associated with the Message Queue Interface (MQI). This central part requires a number of service functions, called *installable services*, in order to perform its work. The installable services are:

- Authorization service
- Name service

Each installable service is a related set of functions implemented using one or more *service components*. Each component is invoked using a properly-architected, publicly-available interface. This enables independent software vendors and other third parties to provide installable components to augment or replace those provided by the WebSphere MQ products. Table 24 summarizes the services and components that can be used on the WebSphere MQ V5.3 platforms.

Table 24. Installable service components summary

Installable service	Supplied component	Function	Requirements
Authorization service	Object Authority Manager (OAM)	Provides authorization checking on commands and MQI calls. Users can write their own component to augment or replace the OAM.	(Appropriate platform authorization facilities are assumed)

Table 24. Installable service components summary (continued)

Installable service	Supplied component	Function	Requirements
Name service	DCE name service component	• Allows queue managers to share queues, or • User defined **Note:** Shared queues must have their *Scope* attribute set to CELL.	• DCE is required for the supplied component, or • A third-party or user-written name manager

Functions and components

Each service consists of a set of related functions. For example, the name service contains function for:

- Looking up a queue name and returning the name of the queue manager where the queue is defined
- Inserting a queue name into the service's directory
- Deleting a queue name from the service's directory

It also contains initialization and termination functions.

An installable service is provided by one or more service components. Each component can perform some or all of the functions that are defined for that service. For example, in WebSphere MQ for AIX, the supplied authorization service component, the OAM, performs all the available functions. See "Authorization service interface" on page 344 for more information. The component is also responsible for managing any underlying resources or software (for example, DCE name services) that it needs to implement the service. Configuration files provide a standard way of loading the component and determining the addresses of the functional routines that it provides.

Figure 29 on page 335 shows how services and components are related:

- A service is defined to a queue manager by stanzas in a configuration file.
- Each service is supported by supplied code in the queue manager. Users cannot change this code and therefore cannot create their own services.
- Each service is implemented by one or more components; these can be supplied with the product or user-written. Multiple components for a service can be invoked, each supporting different facilities within the service.
- Entry points connect the service components to the supporting code in the queue manager.

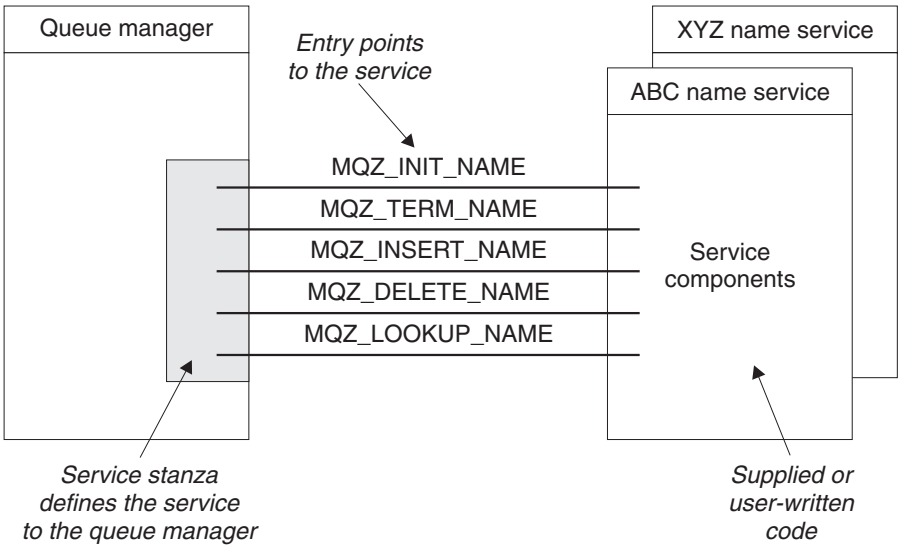


Figure 29. Understanding services, components, and entry points

Entry-points

Each service component is represented by a list of the entry-point addresses of the routines that support a particular installable service. The installable service defines the function to be performed by each routine.

The ordering of the service components when they are configured defines the order in which entry-points are called in an attempt to satisfy a request for the service.

In the supplied header file `cmqzc.h`, the supplied entry points to each service have an MQZID_ prefix.

Return codes

Service components provide return codes to the queue manager to report on a variety of conditions. They report the success or failure of the operation, and indicate whether the queue manager is to proceed to the next service component. A separate *Continuation* parameter carries this indication.

Component data

A single service component might require data to be shared between its various functions. Installable services provide an optional data area to be passed on each invocation of a given service component. This data area is for the exclusive use of the service component. It is shared by all the invocations of a given function, even if they are made from different address spaces or processes. It is guaranteed to be addressable from the service component whenever it is called. You must declare the size of this area in the *`ServiceComponent`* stanza.

Initialization

When the component initialization routine is invoked, it must call the queue manager **MQZEP** function for each entry-point supported by the component. **MQZEP** defines an entry-point to the service. All the undefined exit points are assumed to be NULL.

Primary initialization

A component is always invoked with this option once, before it is invoked in any other way.

Secondary initialization

A component can be invoked with this option on certain platforms. For example, it can be invoked once for each operating system process, thread, or task by which the service is accessed.

If secondary initialization is used:

- The component can be invoked more than once for secondary initialization. For each such call, a matching call for secondary termination is issued when the service is no longer needed.

  For naming services this is the MQZ_TERM_NAME call.

  For authorization services this is the MQZ_TERM_AUTHORITY call.
- The entry points must be re-specified (by calling MQZEP) each time the component is called for primary and secondary initialization.
- Only one copy of component data is used for the component; there is not a different copy for each secondary initialization.
- The component is not invoked for any other calls to the service (from the operating system process, thread, or task, as appropriate) before secondary initialization has been carried out.
- The component must set the *Version* parameter to the same value for primary and secondary initialization.

Primary termination

The primary termination component is always invoked with this option once, when it is no longer required. No further calls are made to this component.

Secondary termination

The secondary termination component is invoked with this option, if it has been invoked for secondary initialization.

Configuring services and components

Configure service components using the queue manager configuration files, except on Windows systems, where each queue manager has its own stanza in the Registry. Each service used must have a *Service* stanza, which defines the service to the queue manager.

For each component within a service, there must be a *ServiceComponent* stanza. This identifies the name and path of the module containing the code for that component.

The authorization service component, known as the Object Authority Manager (OAM), is supplied with the product. When you create a queue manager, the queue manager configuration file (or the Registry on Windows systems) is automatically updated to include the appropriate stanzas for the authorization service and for the default component (the OAM). For the other components, you must configure the queue manager configuration file manually.

The code for each service component is loaded into the queue manager when the queue manager is started, using dynamic binding, where this is supported on the platform.

Service stanza format

The format of the *Service* stanza is:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

where:

<service_name>
The name of the service. This is defined by the service.

<entries>
The number of entry-points defined for the service. This includes the initialization and termination entry points.

Service stanza format for Windows systems

On Windows systems, the *Service* stanza has a third attribute, `SecurityPolicy`. The format of the stanza is:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

where:

<service_name>
The name of the service. This is defined by the service.

<entries>
The number of entry-points defined for the service. This includes the initialization and termination entry points.

<policy>
`NTSIDsRequired` (the Windows Security Identifier), or `Default`. If you do not specify `NTSIDsRequired`, the `Default` value is used. This attribute is valid only if `Name` has a value of `AuthorizationService`.

Service component stanza format

The format of the *Service component* stanza is:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

where:

<service_name>
The name of the service. This must match the `Name` specified in a service stanza.

<component_name>
A descriptive name of the service component. This must be unique, and contain only the characters that are valid for the names of WebSphere MQ objects (for example, queue names). This name occurs in operator messages generated by the service. We recommend that you use a name starting with a company trademark or similar distinguishing string.

<module_name>
The name of the module to contain the code for this component. Specify a full path name.

<size> The size in bytes of the component data area passed to the component on each call. Specify zero if no component data is required.

These two stanzas can appear in any order and the stanza keys under them can also appear in any order. For either of these stanzas, all the stanza keys must be present. If a stanza key is duplicated, the last one is used.

At startup time, the queue manager processes each service component entry in the configuration file in turn. It then loads the specified component module, invoking the entry-point of the component (which must be the entry-point for initialization of the component), passing it a configuration handle.

Creating your own service component

To create your own service component:

- For all platforms, ensure that the header file `cmqzc.h` is included in your program.
- **For UNIX systems**: Create the shared library by compiling the program and linking it with the shared libraries `libmqm*` and `libmqmzf*`. (The threading suffixes and file extensions vary by platform.)

  Note: Because the agent can run in a threaded environment, you must build the OAM and Name Service to run in a threaded environment. This includes using the threaded versions of `libmqm` and `libmqmzf`.
- **For Windows**: Create a DLL by compiling the program, and linking it with the libraries MQM.LIB and MQMZF.LIB.

  See the *WebSphere MQ Application Programming Guide* for details of how to compile and link code for shared libraries.
- Add stanzas to the queue manager configuration file to define the service to the queue manager and to specify the location of the module. Refer to the individual chapters for each service, for more information.
- Stop and restart the queue manager to activate the component.

Using multiple service components

You can install more than one component for a given service. This allows components to provide only partial implementations of the service, and to rely on other components to provide the remaining functions.

Example of using multiple components

Suppose you create a new name service component called `XYZ_name_serv`. This component supports looking up a queue name, but does not support inserting a name in, or deleting a name from, the service directory.

What the component does

The component uses a simple algorithm that returns a fixed queue-manager name for any queue name with which it is invoked. It does not hold a database of queue names, and therefore does not support the insert and delete functions.

How the component is used

The component is installed on the same queue manager as the WebSphere MQ DCE-based name services component. The *`ServiceComponent`* stanzas are ordered so that the DCE-based component is invoked first. Any calls to insert or delete a queue in a component directory are handled by the DCE-based component; it is the only one that implements these functions. However, a lookup call that the DCE-based component cannot resolve is passed on to the lookup-only component, `XYZ_name_serv`. This component supplies a queue-manager name from its simple algorithm.

Omitting entry points when using multiple components

If you decide to use multiple components to provide a service, you can design a service component that does not implement certain functions. The installable services framework places no restrictions on which you can omit. However, for specific installable services, omission of one or more functions might be logically inconsistent with the purpose of the service.

Example of entry points used with multiple components

Table 25 shows an example of the installable name service for which the two components have been installed. Each supports a different set of functions associated with this particular installable service. For insert function, the ABC component entry-point is invoked first. Entry points that have not been defined to the service (using **MQZEP**) are assumed to be NULL. An entry-point for initialization is provided in the table, but this is not required because initialization is carried out by the main entry-point of the component.

When the queue manager has to use an installable service, it uses the entry-points defined for that service (the columns in Table 25). Taking each component in turn, the queue manager determines the address of the routine that implements the required function. It then calls the routine, if it exists. If the operation is successful, any results and status information are used by the queue manager.

Table 25. Example of entry-points for an installable service

Function number	ABC name service component	XYZ name service component
MQZID_INIT_NAME (Initialize)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Terminate)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Insert)	ABC_Insert()	NULL
MQZID_DELETE_NAME (Delete)	ABC_Delete()	NULL

Table 25. Example of entry-points for an installable service (continued)

Function number	ABC name service component	XYZ name service component
MQZID_LOOKUP_NAME (Lookup)	NULL	XYZ_Lookup()

If the routine does not exist, the queue manager repeats this process for the next component in the list. In addition, if the routine does exist but returns a code indicating that it could not perform the operation, the attempt continues with the next available component. Routines in service components might return a code that indicates that no further attempts to perform the operation should be made.

Chapter 20. Authorization service

The authorization service is an installable service that enables queue managers to invoke authorization facilities, for example, checking that a user ID has authority to open a queue.

This service is a component of the WebSphere MQ security enabling interface (SEI), which is part of the WebSphere MQ framework.

This chapter discusses:

Object authority manager (OAM)

The authorization service component supplied with the WebSphere MQ products is called the Object Authority Manager (OAM). By default, the OAM is active and works with the control commands **dspmqaut** (display authority), **dspmqmaut** (display object authority), **dmpmqaut** (dump object authority), **grtmqmaut** (grant object authority), **rvkmqmaut** (revoke object authority), and **setmqaut** (set and reset authority).

The syntax of these commands and how to use them are described in Chapter 17, "The control commands" on page 247.

The OAM works with the *entity* of a principal or group. These entities vary from platform to platform.

When an MQI request is made or a command is issued, the OAM checks the authorization of the entity associated with the operation to see whether it can:
- Perform the requested operation.
- Access the specified queue manager resources.

The authorization service enables you to augment or replace the authority checking provided for queue managers by writing your own authorization service component.

Defining the service to the operating system

The authorization service stanzas in the queue manager configuration file `qm.ini` (or Registry on Windows systems), define the authorization service to the queue manager. See "Configuring services and components" on page 336 for information about the types of stanza.

Refreshing the OAM after changing a user's authorization

In WebSphere MQ, you can update the OAM's authorization group information immediately after changing a user's authorization group membership, reflecting changes made at the operating system level, without needing to stop and restart the queue manager.

Note: When you change authorizations with the **setmqaut** command, the OAM implements such changes immediately.

Queue managers store authorization data on a local queue called SYSTEM.AUTH.DATA.QUEUE. This data is managed by **amqzfuma.exe**.

Migrating from MQSeries

All authorization data is migrated from the authorization files to the authorization queue the first time that you restart the queue manager after migrating from MQSeries. If the OAM detects a missing file:

1. If the authorization applies to a single object, the OAM gives the mqm group (or Administrator on Windows systems) access to the object and continues with the migration. Message AMQ5528 is written to the queue manager's error log. Refer to *WebSphere MQ Messages* for more information about message AMQ5528.
2. If the authorization applies to a class of objects, the OAM stops the migration. The queue manager does not start until the file has been replaced.

Continuing to store authorization data in files

You can continue to store authorization data in files, but this affects the performance of the OAM. Storing authorization data on a local queue reduces the time needed to check an authorization. Furthermore, if you continue to store your authorization data in files, you **cannot** use generic profiles, limiting the flexibility with which you can apply authorizations. For information on the uses of generic profiles, see "Using OAM generic profiles" on page 123.

The default OAM service module is amqzfu on UNIX systems, or amqzfu.dll on Windows systems. WebSphere MQ also provides the previous service module as amqzfu0 (amqzfu0.dll). There are two ways in which you can use the previous module to continue to store authorization data in files:

1. Modify the `Module` attribute in the `ServiceComponent` stanza of the qm.ini file (Registry entry on Windows systems) to use amqzfu0 (amqzfu0.dll). This option is possible only for queue managers created with a version of MQSeries before Version 5.2.
2. Replace the amqzfu (amqzfu.dll) module by the previous version by:
   a. Removing the *new* amqzfu module
   b. Renaming amqzfu0 as amqzfu

   You can restore the new amqzfu module from the copy provided as amqzfu1 (amqzfu1.dll).

Once you have created or restarted a queue manager with the new amqzfu module, you can no longer replace it with the previous version. The migration process is not reversible.

Authorization service on UNIX systems

On these platforms:

Principal
Is a UNIX system user ID, or an ID associated with an application program running on behalf of a user.

Group Is a UNIX system-defined collection of principals.

Authorizations can be granted or revoked at the group level only. A request to grant or revoke a user's authority updates the primary group for that user.

Configuring authorization service stanzas: UNIX systems

On UNIX systems, each queue manager has its own queue manager configuration file.

For example, on AIX, the default path and file name of the queue manager configuration file for queue manager QMNAME is `/var/mqm/qmgrs/QMNAME/qm.ini`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden by `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

For example, the following stanzas in the queue manager configuration file define two authorization service components on WebSphere MQ for AIX:

```
Service:
   Name=AuthorizationService
   EntryPoints=7

ServiceComponent:
   Service=AuthorizationService
   Name=MQ.UNIX.authorization.service
   Module=/usr/mqm/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

Figure 30. UNIX authorization service stanzas in qm.ini

The service component stanza (`MQ.UNIX.authorization.service`) defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service on Windows systems

On Windows systems:

Principal
Is a Windows user ID, or an ID associated with an application program running on behalf of a user.

Group Is a Windows group.

Authorizations can be granted or revoked at the principal or group level.

Configuring authorization service stanzas: Windows systems

On WebSphere MQ for Windows each queue manager has its own stanza in the registry.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to the Registry automatically, but can be overridden using `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

You can also add the `SecurityPolicy` attribute using the WebSphere MQ services. The `SsecurityPolicy` attribute applies only if the service specified on the *Service*

Windows systems

stanza is the authorization service, that is, the default OAM. The `SecurityPolicy` attribute allows you to specify the security policy for each queue manager. The possible values are:

`Default`
Specify `Default` if you want the default security policy to take effect. If a Windows security identifier (NT SID) is not passed to the OAM for a particular user ID, an attempt is made to obtain the appropriate SID by searching the relevant security databases.

`NTSIDsRequired`
Requires that an NT SID is passed to the OAM when performing security checks.

For more general information about security, see Chapter 10, "WebSphere MQ security" on page 113.

The service component stanza, `MQSeries.WindowsNT.auth.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service interface

The authorization service provides the following entry points for use by the queue manager:

MQZ_INIT_AUTHORITY
Initializes authorization service component.

MQZ_TERM_AUTHORITY
Terminates authorization service component.

MQZ_CHECK_AUTHORITY
Checks whether an entity has authority to perform one or more operations on a specified object.

MQZ_SET_AUTHORITY
Sets the authority that an entity has to a specified object.

MQZ_GET_AUTHORITY
Gets the authority that an entity has to access a specified object.

MQZ_GET_EXPLICIT_AUTHORITY
Gets either the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group) or the authority that the primary group of the named principal has to access a specified object.

MQZ_COPY_ALL_AUTHORITY
Copies all the current authorizations that exist for a referenced object to another object.

MQZ_ENUMERATE_AUTHORITY_DATA
Retrieves all the authority data that matches the selection criteria specified.

MQZ_DELETE_AUTHORITY
Deletes all authorizations associated with a specified object.

MQZ_REFRESH_CACHE
Refresh all authorizations.

In addition, on WebSphere MQ for Windows, the authorization service provides the following entry points for use by the queue manager:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**

These entry points support the use of the Windows Security Identifier (NT SID).

These names are defined as **typedef**s, in the header file `cmqzc.h`, which can be used to prototype the component functions.

The initialization function (**MQZ_INIT_AUTHORITY**) must be the main entry point for the component. The other functions are invoked through the entry point address that the initialization function has added into the component entry point vector.

See "Creating your own service component" on page 338 for more information.

Chapter 21. Name service

The name service is an installable service that provides support to the queue manager for looking up the name of the queue manager that owns a specified queue. No other queue attributes can be retrieved from a name service.

The name service enables an application to open remote queues for output as if they were local queues. A name service is not invoked for objects other than queues.

Note: The remote queues ***must*** have their *Scope* attribute set to CELL.

When an application opens a queue, it looks for the name of the queue first in the queue manager's directory. If it does not find it there, it looks in as many name services as have been configured, until it finds one that recognizes the queue name. If none recognizes the name, the open fails.

The name service returns the owning queue manager for that queue. The queue manager then continues with the MQOPEN request as if the command had specified the queue and queue manager name in the original request.

The name service interface (NSI) is part of the WebSphere MQ framework.

This chapter discusses:

How the name service works

If a queue definition specifies the *Scope* attribute as queue manager, that is, SCOPE(QMGR) in MQSC, the queue definition (along with all the queue attributes) is stored in the queue manager's directory only. This cannot be replaced by an installable service.

If a queue definition specifies the *Scope* attribute as cell, that is, SCOPE(CELL) in MQSC, the queue definition is again stored in the queue manager's directory, along with all the queue attributes. However, the queue and queue-manager name are also stored in a name service. If no service is available that can store this information, a queue with the *Scope* cell cannot be defined.

The directory in which the information is stored can be managed by the service, or the service can use an underlying service, for example, a DCE directory, for this purpose. In either case, definitions stored in the directory must persist, even after the component and queue manager have terminated, until they are explicitly deleted.

Notes:

1. To send a message to a remote host's local queue definition (with a scope of CELL) on a different queue manager within a naming directory cell, you need to define a channel.
2. You cannot get messages directly from the remote queue, even when it has a scope of CELL.

3. No remote queue definition is required when sending to a queue with a scope of CELL.
4. The naming service centrally defines the destination queue, although you still need a transmission queue to the destination queue manager and a pair of channel definitions. In addition, the transmission queue on the local system must have the same name as the queue manager owning the target queue, with the scope of cell, on the remote system.

   For example, if the remote queue manager has the name QM01, the transmission queue on the local system must also have the name QM01. See *WebSphere MQ Intercommunication* for further information.

Name service interface

A name service provides the following entry points for use by the queue manager:

MQZ_INIT_NAME
Initialize the name service component.

MQZ_TERM_NAME
Terminate the name service component.

MQZ_LOOKUP_NAME
Look up the queue-manager name for the specified queue.

MQZ_INSERT_NAME
Insert an entry containing the owning queue-manager name for the specified queue into the directory used by the service.

MQZ_DELETE_NAME
Delete the entry for the specified queue from the directory used by the service.

If there is more than one name service configured:

- For lookup, the MQZ_LOOKUP_NAME function is invoked for each service in the list until the queue name is resolved (unless any component indicates that the search should stop).
- For insert, the MQZ_INSERT_NAME function is invoked for the first service in the list that supports this function.
- For delete, the MQZ_DELETE_NAME function is invoked for the first service in the list that supports this function.

Do not have more than one component that supports the insert and delete functions. However, a component that only supports lookup is feasible, and could be used, for example, as the last component in the list to resolve any name that is not known by any other name service component to a queue manager at which the name can be defined.

In the C programming language the names are defined as function datatypes using the typedef statement. These can be used to prototype the service functions, to ensure that the parameters are correct.

The header file that contains all the material specific to installable services is `cmqzc.h` for the C language.

Apart from the initialization function (MQZ_INIT_NAME), which must be the component's main entry point, functions are invoked by the entry point address that the initialization function has added, using the MQZEP call.

The following examples of UNIX configuration file stanzas for the name service specify a name service component provided by the (fictitious) ABC company.

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

Figure 31. Name service stanzas in qm.ini (for UNIX systems)

Note: On Windows systems, name service stanza information is stored in the Registry.

Using DCE to share queues on different queue managers

We supply an implementation of a name service that uses DCE (Distributed Computing Environment), although you are free to write your own component that does not use DCE.

To use the supplied name service component, define the name service and its installed component to the queue manager by inserting the appropriate stanza in the queue manager configuration file (`qm.ini`), or the Registry for Windows systems. See "Installable services" on page 100 for information on how to do this. You must also configure DCE as described in "DCE configuration" on page 350.

Note: You cannot use the IBM-supplied DCE name service with WebSphere MQ Secure Sockets Layer (SSL) software.

If your queue managers are located on nodes within a Distributed Computing Environment (DCE) cell, you can configure them to share queues. Applications can then connect to one queue manager and open a queue for output on another queue manager on another node.

The queue manager rejects open requests from a local application if the queue is not defined on that queue manager. However, when the DCE name service is in use, the remote queue does not need to be defined on the local queue manager. Also, if an appropriate set of transmission queues are defined, the queue can be moved between remote queue managers within the DCE cell without any changes being required to the local definitions.

Configuration tasks for shared queues

This section describes how you set up shared queues on queue managers that reside on nodes that are within the DCE cell.

For each queue manager:

1. Use the **endmqm** command to stop the queue manager if it is running.
2. Configure the name service by adding the required name service stanza to the queue manager configuration file. To invoke the name service, you have to restart the queue manager.

3. Use the **strmqm** command to restart the queue manager.
4. Set up channels for messaging between queue managers; see *WebSphere MQ Intercommunication* for further details.

For any queue that you want to be shared, specify the SCOPE attribute as CELL. For example, use these MQSC commands:

```
DEFINE QLOCAL (GREY.PUBLIC.QUEUE) SCOPE(CELL)
or
ALTER QLOCAL (PINK.LOCAL.QUEUE) SCOPE(CELL)
```

The queue created or altered must belong to a queue manager on a node within the DCE cell.

DCE configuration

To use the supplied name service component, you must have the OSF Distributed Computing Environment (DCE) Directory Service configured. This service enables applications that connect to one queue manager to open queues that belong to another queue manager in the same DCE cell.

You can configure WebSphere MQ to use the DCE name service as follows:

1. Ensure that DCE is started. To do this, on the command line type:

   ```
   $ dcecp
   dcecp> host catalog
   ```

   You are presented with a list of hosts in the cell. This indicates that DCE is started.

   Note: If DCE is not running, you see the following error message:

   ```
   error with socket
   ```

2. Log in to DCE as `cell_admin` and run the sample programs `dcesetsv` and `dcesetkp` that define `mqm` and create a set of directory entries so that the supplied name service can run.

   Both sample programs are located in the following directories:

   For WebSphere MQ for Windows

   `C:\MQM\TOOLS\DCE\SAMPLES`, where `C` is the installation drive.

   For UNIX systems

   `/usr/mqm/samp` on AIX.

   `/opt/mqm/samp` on HP-UX, Solaris, and Linux.

> **Note to users**
>
> You need to install the MQ DCE option if you are using HP-UX, AIX, or Solaris.

To run sample program `dcesetsv`, use the following command:

```
dcesetsv <mqm's password> <cell_admin's password>
```

This program sets up `mqm` as a principal to DCE, creates a set of directory entries for `mqm`, and gives it access rights to the DCE cell servers. The program can be run from any host in the cell, and is run ***once*** for the entire DCE cell.

3. For each host in the DCE cell that will be using the DCE name service, run sample program `dcesetkt` using the following command:

   ```
   dcesetkt <mqm's password>
   ```

   This program sets up a system keytable file on the local host that contains information about principal `mqm` and its password.

   The keytable files are located in the following directories:

   For WebSphere MQ for Windows
   `C:\MQM\DCE\KEYTABS\KEYTAB`, where `C` is the installation drive.

   For UNIX systems
   `/var/mqm/dce/keytabs/keytab`

   Note: When the keytable is first created on UNIX systems, it does not have the correct permissions to allow the DCE naming service to be used. Ensure that the correct permissions exist by issuing the following commands from the UNIX command line:

   ```
   $ cd /var/mqm/dce/keytabs
   $ chown mqm:mqm keytab
   $ chmod u+rw keytab
   $ chmod g+r keytab
   ```

DCE configuration

Chapter 22. Installable services interface reference information

This chapter provides reference information for the installable services. It includes:

The functions and data types are in alphabetic order within the group for each service type.

Table 26. Installable services functions

Service type	*Functions*	*page*
All	MQZEP – Add component entry point MQHCONFIG – Configuration handle PMQFUNC – Pointer to function	355 356 356
Authorization	MQZ_CHECK_AUTHORITY MQZ_COPY_ALL_AUTHORITY MQZ_DELETE_AUTHORITY MQZ_ENUMERATE_AUTHORITY_DATA MQZ_GET_AUTHORITY MQZ_GET_EXPLICIT_AUTHORITY MQZ_INIT_AUTHORITY MQZ_REFRESH_CACHE MQZ_SET_AUTHORITY MQZ_TERM_AUTHORITY	357 367 370 373 376 382 388 391 393 399
Extended authorization	MQZ_CHECK_AUTHORITY_2 MQZ_GET_AUTHORITY_2 MQZ_GET_EXPLICIT_AUTHORITY_2 MQZ_SET_AUTHORITY_2	362 379 385 396

Table 26. Installable services functions (continued)

Service type	Functions	page
Name	MQZ_DELETE_NAME	406
	MQZ_INIT_NAME	408
	MQZ_INSERT_NAME	411
	MQZ_LOOKUP_NAME	413
	MQZ_TERM_NAME	416

How the functions are shown

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in the order they must occur. They must all be present.

Parameters and data types

Each parameter name is followed by its data type in parentheses. These are the elementary data types described in the *WebSphere MQ Application Programming Reference* manual.

The C language invocation is also given, after the description of the parameters.

MQZEP – Add component entry point

This function is invoked by a service component, during initialization, to add an entry point to the entry point vector for that service component.

Syntax

MQZEP *(Hconfig, Function, EntryPoint, CompCode, Reason)*

Parameters

The MQZEP call has the following parameters.

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the component which is being configured for this particular installable service. It must be the same as the one passed to the component configuration function by the queue manager on the component initialization call.

Function (MQLONG) – input
Function identifier.

Valid values for this are defined for each installable service.

If MQZEP is called more than once for the same function, the last call made provides the entry point which is used.

EntryPoint (PMQFUNC) – input
Function entry point.

This is the address of the entry point provided by the component to perform the function.

The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points which are not defined using MQZEP.

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_FUNCTION_ERROR
(2281, X'8E9') Function identifier not valid.
MQRC_HCONFIG_ERROR
(2280, X'8E8') Configuration handle not valid.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;     /* Configuration handle */
MQLONG     Function;    /* Function identifier */
PMQFUNC    EntryPoint;  /* Function entry point */
MQLONG     CompCode;    /* Completion code */
MQLONG     Reason;      /* Reason code qualifying CompCode */
```

MQHCONFIG – Configuration handle

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

C declaration

```
typedef void MQPOINTER MQHCONFIG;
```

PMQFUNC – Pointer to function

Pointer to a function.

C declaration

```
typedef void MQPOINTER PMQFUNC;
```

MQZ_CHECK_AUTHORITY – Check authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

Syntax

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_CHECK_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. It is one of the following:

MQZAET_PRINCIPAL
: Principal.

MQZAET_GROUP
: Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations described above.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_ALL_ADMIN
All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY_2 – Check authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

MQZ_CHECK_AUTHORITY_2 is similar to MQZ_CHECK_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_CHECK_AUTHORITY_2 (QMgrName, EntityData, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_CHECK_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose authorization to the object is to be checked. See "MQZED – Entity descriptor" on page 404 for details.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. It is one of the following:

MQZAET_PRINCIPAL
: Principal.

MQZAET_GROUP
: Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations described above.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_ALL_ADMIN
All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY_2 this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_CHECK_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                       ObjectName, ObjectType, Authority, ComponentData,
                       &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY – Copy all authority

This function is provided by an authorization service component. It is invoked by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

Syntax

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName,
        ObjectType, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_COPY_ALL_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

RefObjectName (MQCHAR48) – input

Reference object name.

The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType (MQLONG) – input

Object type.

The type of object specified by *RefObjectName* and *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
: Authentication information.

MQOT_NAMELIST
: Namelist.

MQOT_PROCESS
: Process definition.

MQOT_Q
: Queue.

MQOT_Q_MGR
Queue manager.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_COPY_ALL_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_REF_OBJECT
(2294, X'8F6') Reference object unknown.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,
                        ComponentData, &Continuation, &CompCode,
                        &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  RefObjectName;     /* Reference object name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY – Delete authority

This function is provided by an authorization service component, and is invoked by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

Syntax

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType,
        ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_DELETE_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_DELETE_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
```

```
                                   component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data

This function is provided by an MQZAS_VERSION_4 authorization service component, and is invoked repeatedly by the queue manager to retrieve all of the authority data that matches the selection criteria specified on the first invocation.

The function identifier for this function (for MQZEP) is MQZID_ENUMERATE_AUTHORITY_DATA.

Syntax

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration,
        Filter, AuthorityBufferLength, AuthorityBuffer, AuthorityDataLength,
        ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_ENUMERATE_AUTHORITY_DATA call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

StartEnumeration (MQLONG) – input

Flag indicating whether call should start enumeration.

This indicates whether the call should start the enumeration of authority data, or continue the enumeration of authority data started by a previous call to MQZ_ENUMERATE_AUTHORITY_DATA. The value is one of the following:

MQZSE_START
Start enumeration.

The call is invoked with this value to start the enumeration of authority data. The *Filter* parameter specifies the selection criteria to be used to select the authority data returned by this and successive calls.

MQZSE_CONTINUE
Continue enumeration.

The call is invoked with this value to continue the enumeration of authority data. The *Filter* parameter is ignored in this case, and can be specified as the null pointer (the selection criteria are determined by the *Filter* parameter specified by the call that had *StartEnumeration* set to MQZSE_START).

Filter (MQZAD) – input

Filter.

If *StartEnumeration* is MQZSE_START, *Filter* specifies the selection criteria to be used to select the authority data to return. If *Filter* is the null pointer, no selection

criteria are used, that is, all authority data is returned. See "MQZAD – Authority data" on page 401 for details of the selection criteria that can be used.

If *StartEnumeration* is MQZSE_CONTINUE, *Filter* is ignored, and can be specified as the null pointer.

AuthorityBufferLength (MQLONG) – input

Length of *AuthorityBuffer*.

This is the length in bytes of the *AuthorityBuffer* parameter. The authority buffer must be big enough to accommodate the data to be returned.

AuthorityBuffer (MQZAD) – output

Authority data.

This is the buffer in which the authority data is returned. The buffer must be big enough to accommodate an MQZAD structure, an MQZED structure, plus the longest entity name and longest domain name defined.

Note: This parameter is defined as an MQZAD, as the MQZAD always occurs at the start of the buffer. However, if the buffer is actually declared as an MQZAD, the buffer will be too small – it needs to be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, plus entity and domain names.

AuthorityDataLength (MQLONG) – output

Length of data returned in *AuthorityBuffer*.

This is the length of the data returned in *AuthorityBuffer*. If the authority buffer is too small, *AuthorityDataLength* is set to the length of the buffer required, and the call returns completion code MQCC_FAILED and reason code MQRC_BUFFER_LENGTH_ERROR.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.
MQRC_NO_DATA_AVAILABLE
(2379, X'94B') No data available.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration, &Filter,
                              AuthorityBufferLength,
                              &AuthorityBuffer,
                              &AuthorityDataLength, ComponentData,
                              &Continuation, &CompCode,
                              &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;               /* Queue manager name */
MQLONG    StartEnumeration;       /* Flag indicating whether call should
                                     start enumeration */
MQZAD     Filter;                 /* Filter */
MQLONG    AuthorityBufferLength;  /* Length of AuthorityBuffer */
MQZAD     AuthorityBuffer;        /* Authority data */
MQLONG    AuthorityDataLength;    /* Length of data returned in
                                     AuthorityBuffer */
MQBYTE    ComponentData[n];       /* Component data */
MQLONG    Continuation;           /* Continuation indicator set by
                                     component */
MQLONG    CompCode;               /* Completion code */
MQLONG    Reason;                 /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY – Get authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

Syntax

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
        ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_GET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, &Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY_2 – Get authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

MQZ_GET_AUTHORITY_2 is similar to MQZ_GET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_GET_AUTHORITY_2 (QMgrName, EntityData, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_GET_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be retrieved. See "MQZED – Entity descriptor" on page 404 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY_2 this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, &Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_GET_EXPLICIT_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
: Principal.

MQZAET_GROUP
: Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_EXPLICIT_AUTHORITY this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, &Authority,
                            ComponentData, &Continuation,
                            &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY_2 – Get explicit authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

MQZ_GET_EXPLICIT_AUTHORITY_2 is similar to MQZ_GET_EXPLICIT_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, EntityData, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_GET_EXPLICIT_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be retrieved. See "MQZED – Entity descriptor" on page 404 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_EXPLICIT_AUTHORITY_2 this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                              ObjectName, ObjectType, &Authority,
                              ComponentData, &Continuation,
                              &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY – Initialize authorization service

This function is provided by an authorization service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

Syntax

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
        ComponentData, Version, CompCode, Reason)
```

Parameters

The MQZ_INIT_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input

Initialization options.

It is one of the following:

MQZIO_PRIMARY
: Primary initialization.

MQZIO_SECONDARY
: Secondary initialization.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input

Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the

initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Version (MQLONG) – input/output

Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following values are suppported:

MQZAS_VERSION_1
Version 1.

MQZAS_VERSION_2
Version 2.

MQZAS_VERSION_3
Version 3.

MQZAS_VERSION_4
Version 4.

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_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, &Version, &CompCode,
                    &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;             /* Configuration handle */
MQLONG     Options;             /* Initialization options */
MQCHAR48   QMgrName;            /* Queue manager name */
```

```
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_REFRESH_CACHE – Refresh all authorizations

This function is provided by an MQZAS_VERSION_3 authorization service component, and is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax

MQZ_REFRESH_CACHE
(QMgrName, ComponentData, Continuation, CompCode, Reason)

Parameters

QMgrName (MQCHAR48) — input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) — input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) — output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_REFRESH_CACHE this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

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_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information on this reason code, see the *WebSphere MQ Application Programming Reference* book.

C invocation

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY – Set authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

Syntax

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
        ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_SET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be set. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_SET_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY_2 – Set authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

MQZ_SET_AUTHORITY_2 is similar to MQZ_SET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_SET_AUTHORITY_2 (QMgrName, EntityData, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
        Reason)
```

Parameters

The MQZ_SET_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be set. See "MQZED – Entity descriptor" on page 404 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input
Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_SET_AUTHORITY_2 this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

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_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_SET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY – Terminate authorization service

This function is provided by an authorization service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

Syntax

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
        CompCode, Reason)
```

Parameters

The MQZ_TERM_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input

Termination options.

It is one of the following:

MQZTO_PRIMARY
: Primary termination.

MQZTO_SECONDARY
: Secondary termination.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

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_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZAD – Authority data

The following table summarizes the fields in the structure.

Table 27. Fields in MQZAD

Field	Description	Page
StrucId	Structure identifier	401
Version	Structure version number	401
ProfileName	Profile name	402
ObjectType	Object type	402
Authority	Authority	402
EntityDataPtr	Address of MQZED structure identifying an entity	402
EntityType	Type of entity	402

The MQZAD structure is used on the MQZ_ENUMERATE_AUTHORITY_DATA call for two parameters:

- MQZAD is used for the *Filter* parameter which is input to the call. This parameter specifies the selection criteria that are to be used to select the authority data returned by the call.
- MQZAD is also used for the *AuthorityBuffer* parameter which is output from the call. This parameter specifies the authorizations for one combination of profile name, object type, and entity.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZAD_STRUC_ID
Identifier for authority data structure.

For the C programming language, the constant MQZAD_STRUC_ID_ARRAY is also defined; this has the same value as MQZAD_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZAD_VERSION_1
Version-1 authority data structure.

The following constant specifies the version number of the current version:

MQZAD_CURRENT_VERSION
Current version of authority data structure.

This is an input field to the service.

ProfileName (MQCHAR48)
Profile name.

For the *Filter* parameter, this field is the profile name whose authority data is required. If the name is entirely blank up to the end of the field or the first null character, authority data for all profile names is returned.

For the *AuthorityBuffer* parameter, this field is the name of a profile that matches the specified selection criteria.

ObjectType (MQLONG)
Object type.

For the *Filter* parameter, this field is the object type for which authority data is required. If the value is MQOT_ALL, authority data for all object types is returned.

For the *AuthorityBuffer* parameter, this field is the object type to which the profile identified by *ProfileName* applies.

The value is one of the following; for the *Filter* parameter, the value MQOT_ALL is also valid:

MQOT_Q
Queue.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q_MGR
Queue manager.

MQOT_AUTH_INFO
Authentication information.

Authority (MQLONG)
Authority.

For the *Filter* parameter, this field is ignored.

For the *AuthorityBuffer* parameter, this field represents the authorizations that the entity has to the objects identified by *ProfileName* and *ObjectType*. If the entity has only one authority, the field is equal to the appropriate authorization value (MQZAO_* constant). If the entity has more than one authority, the field is the bitwise OR of the corresponding MQZAO_* constants.

EntityDataPtr (PMQZED)
Address of MQZED structure identifying an entity.

For the *Filter* parameter, this field points to an MQZED structure that identifies the entity whose authority data is required. If *EntityDataPtr* is the null pointer, authority data for all entities is returned.

For the *AuthorityBuffer* parameter, this field points to an MQZED structure that identifies the entity whose authority data has been returned.

EntityType (MQLONG)
Entity type.

For the *Filter* parameter, this field specifies the entity type for which authority data is required. If the value is MQZAET_NONE, authority data for all entity types is returned.

For the *AuthorityBuffer* parameter, this field specifies the type of the entity identified by the MQZED structure pointed to by *EntityDataPtr*.

The value is one of the following; for the *Filter* parameter, the value MQZAET_NONE is also valid:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

C declaration

```
typedef struct tagMQZAD MQZAD;
struct tagMQZAD {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQCHAR48  ProfileName;    /* Profile name */
  MQLONG    ObjectType;     /* Object type */
  MQLONG    Authority;      /* Authority */
  PMQZED    EntityDataPtr;  /* Address of MQZED structure identifying an
                               entity */
  MQLONG    EntityType;     /* Entity type */
};
```

MQZED – Entity descriptor

The following table summarizes the fields in the structure.

Table 28. Fields in MQZED

Field	Description	Page
StrucId	Structure identifier	404
Version	Structure version number	404
EntityNamePtr	Address of entity name	404
EntityDomainPtr	Address of entity domain name	404
SecurityId	Security identifier	405

The MQZED structure describes the information that is passed to the MQZAS_VERSION_2 authorization service calls.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZED_STRUC_ID
Identifier for entity descriptor structure.

For the C programming language, the constant MQZED_STRUC_ID_ARRAY is also defined; this has the same value as MQZED_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)
Structure version number.

The value is:

MQZED_VERSION_1
Version-1 entity descriptor structure.

The following constant specifies the version number of the current version:

MQZED_CURRENT_VERSION
Current version of entity descriptor structure.

This is an input field to the service.

EntityNamePtr (PMQCHAR)
Address of entity name.

This is a pointer to the name of the entity whose authorization is to be checked.

EntityDomainPtr (PMQCHAR)
Address of entity domain name.

This is a pointer to the name of the domain containing the definition of the entity whose authorization is to be checked.

SecurityId (MQBYTE40)

Security identifier.

This is the security identifier whose authorization is to be checked.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  PMQCHAR   EntityNamePtr;    /* Address of entity name */
  PMQCHAR   EntityDomainPtr;  /* Address of entity domain name */
  MQBYTE40  SecurityId;       /* Security identifier */
};
```

MQZ_DELETE_NAME – Delete name

This function is provided by a name service component, and is invoked by the queue manager to delete an entry for the specified queue.

The function identifier for this function (for MQZEP) is MQZID_DELETE_NAME.

Syntax

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, Continuation,
        CompCode, Reason)
```

Parameters

The MQZ_DELETE_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue for which an entry is to be deleted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_DELETE_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT
: Continuation dependent on queue manager.

MQZCI_STOP
: Do not continue with next component.

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_UNKNOWN_Q_NAME
(2288, X'8F0') Queue name not found.

Note: It may not be possible to return this code if the underlying service simply responds with success for this case.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, &Continuation,
                 &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_NAME – Initialize name service

This function is provided by a name service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_NAME.

Syntax

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
        ComponentData, Version, CompCode, Reason)
```

Parameters

The MQZ_INIT_NAME call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input

Initialization options.

It is one of the following:

MQZIO_PRIMARY
: Primary initialization.

MQZIO_SECONDARY
: Secondary initialization.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input

Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the

initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes. Therefore primary initialization is the first process initialization and secondary initialization is any subsequent process initialization.

Version (MQLONG) – input/output

Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_NAME function and makes no further use of this component.

The following value is suppported:

MQZNS_VERSION_1
Version 1.

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_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
               ComponentData, &Version, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
```

MQZ_INIT_NAME

```
MQLONG     Version;             /* Version number */
MQLONG     CompCode;            /* Completion code */
MQLONG     Reason;              /* Reason code qualifying CompCode */
```

MQZ_INSERT_NAME – Insert name

This function is provided by a name service component, and is invoked by the queue manager to insert an entry for the specified queue, containing the name of the queue manager that owns the queue. If the queue is already defined in the service, the call fails.

The function identifier for this function (for MQZEP) is MQZID_INSERT_NAME.

Syntax

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
        Continuation, CompCode, Reason)
```

Parameters

The MQZ_INSERT_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue for which an entry is to be inserted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – input

Resolved queue manager name.

The name of the queue manager to which the queue resolves. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_INSERT_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_STOP
Do not continue with next component.

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_Q_ALREADY_EXISTS
(2290, X'8F2') Queue object already exists.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_LOOKUP_NAME – Lookup name

This function is provided by a name service component, and is invoked by the queue manager to retrieve the name of the owning queue manager, for a specified queue.

The function identifier for this function (for MQZEP) is MQZID_LOOKUP_NAME.

Syntax

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
        Continuation, CompCode, Reason)
```

Parameters

The MQZ_LOOKUP_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue which is to be resolved. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – output

Resolved queue manager name.

If the function completes successfully, this is the name of the queue manager that owns the queue.

The name returned by the service component must be padded on the right with blanks to the full length of the parameter; the name *must not* be terminated by a null character, or contain leading or embedded blanks.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_LOOKUP_NAME, the queue manager decides whether to invoke another name service component, as follows:

- If *CompCode* is MQCC_OK, no further components are invoked, whatever value is returned in *Continuation*.
- If *CompCode* is not MQCC_OK, a further component is invoked, unless *Continuation* is MQZCI_STOP. This value should not be set without good reason.

The following values can be specified:

MQZCI_DEFAULT
: Continuation dependent on queue manager.

MQZCI_CONTINUE
: Continue with next component.

MQZCI_STOP
: Do not continue with next component.

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_SERVICE_ERROR
: (2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
: (2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_Q_NAME
: (2288, X'8F0') Queue name not found.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
```

```
                                   component */
MQLONG    CompCode;               /* Completion code */
MQLONG    Reason;                 /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME – Terminate name service

This function is provided by a name service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_NAME.

Syntax

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData,
        CompCode, Reason)
```

Parameters

The MQZ_TERM_NAME call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input

Termination options.

It is one of the following:

MQZTO_PRIMARY
Primary termination.

MQZTO_SECONDARY
Secondary termination.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_NAME call.

When the MQZ_TERM_NAME call has completed, the queue manager discards this data.

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_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, &CompCode,
               &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME

Chapter 23. 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 use API exits"
- "How you use API exits"
- "What happens when an API exit runs?" on page 421
- "Configuring API exits" on page 421

We give a brief introduction to writing API exits in "How to write an API exit" on page 420. For detailed information about writing API exits, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

Why 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.

How you use API exits

This section gives a brief overview of the tasks involved in setting up API exits.

How to configure WebSphere MQ for API exits

You configure WebSphere MQ to enable API exits by changing the configuration information in the usual ways:

- On WebSphere MQ for Windows, use the WebSphere MQ Services snap-in or the **amqmdain** command to make changes to attribute information within the Windows Registry
- On WebSphere MQ for UNIX systems, edit the WebSphere MQ configuration files, mqs.ini and qm.ini,

In either case, you provide information to:

- Name the API exit
- Identify the module and entry point of the API exit code to run
- Optionally pass data with the exit
- Identify the sequence of this exit in relation to other exits

For detailed information on this configuration, see "Configuring API exits" on page 421. For a description of how API exits run, see "What happens when an API exit runs?" on page 421.

How to write an API exit

This section introduces writing API exits. For detailed information, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

You write your exits using the C programming language. To help you do so, we provide a sample exit, amqsaxe0, that generates trace entries to a named file. When you start writing exits, we recommend that you use this as your starting point.

Exits are available for every API call, as follows:

- MQCONN/MQCONNX, to provide a queue manager connection handle for use on subsequent API calls
- MQDISC, to disconnect from a queue manager
- MQBEGIN, to begin a global unit of work (UOW)
- MQBACK, to back out a UOW
- MQCMIT, to commit a UOW
- MQOPEN, to open an MQSeries resource for subsequent access
- MQCLOSE, to close an MQSeries resource that had previously been opened for access
- MQGET, to retrieve a message from a queue that has previously been opened for access
- MQPUT1, to place a message on to a queue
- MQPUT, to place a message on to a queue that has previously been opened for access
- MQINQ, to inquire on the attributes of an MQSeries resource that has previously been opened for access
- MQSET, to set the attributes of a queue that has previously been opened for access

Within API exits, these calls 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).

What happens when an API exit runs?

The API exit routines to run are identified in stanzas (UNIX) or Registry entries (Windows). (For simplicity, we talk about the stanzas in mqs.ini and qm.ini; Windows users provide the same information using the WebSphere MQ Services snap-in.) The definition of the routines can occur in three places:

1. ApiExitCommon, in the mqs.ini file, identifies routines, for the whole of WebSphere MQ, applied when queue managers start up. These can be overridden by routines defined for individual queue managers.
2. ApiExitTemplate, in the mqs.ini file, identifies routines, for the whole of WebSphere MQ, copied to the ApiExitLocal set when a new queue manager is created.
3. ApiExitLocal, in the qm.ini file, identifies routines applicable to a particular queue manager.

When a new queue manager is created, the ApiExitTemplate definitions in mqs.ini are copied to the ApiExitLocal definitions in qm.ini for the new queue manager. When a queue manager is started, both the ApiExitCommon and ApiExitLocal definitions are used. The ApiExitLocal definitions replace the ApiExitCommon definitions if both identify a routine of the same name. The `Sequence` attribute, described in "Attributes for all stanzas" determines the order in which the routines defined in the stanzas run.

Configuring API exits

This section tells you how to configure API exits. We start in "Configuring API exits on UNIX systems", explaining how to add the stanzas, followed by "Configuring API exits on Windows systems" on page 423, which tells you how to use the WebSphere MQ Services snap-in.

Configuring API exits on UNIX systems

You define your API exits in new stanzas in the mqs.ini and qm.ini files. The sections below describe these stanzas, and the attributes within them that define the exit routines and the sequence in which they run. For guidance on the process of changing these stanzas, see "Changing the configuration information" on page 423.

Stanzas in mqs.ini are:

ApiExitCommon
When any queue manager starts, the attributes in this stanza are read, and then overridden by the API exits defined in qm.ini.

ApiExitTemplate
When any queue manager is created, the attributes in this stanza are copied into the newly created qm.ini file under the ApiExitLocal stanza.

The stanza in qm.ini is:

ApiExitLocal
When the queue manager starts, API exits defined here override the defaults defined in mqs.ini.

Attributes for all stanzas

All these stanzas have the following attributes:

Name=ApiExit_name
The descriptive name of the API exit passed to it in the ExitInfoName field of the MQAXP structure.

This name must be unique, no longer than 48 characters, and contain only valid characters for the names of WebSphere MQ objects (for example, queue names).

Function=function_name
The name of the function entry point into the module containing the API exit code. This entry point is the MQ_INIT_EXIT function.

The length of this field is limited to MQ_EXIT_NAME_LENGTH.

Module=module_name
The module containing the API exit code.

If this field contains the full path name of the module it is used as is.

If this field contains just the module name, the module is located using the `ExitsDefaultPath` attribute in the `ExitPath` in qm.ini.

On platforms that support separate threaded libraries (AIX, HP/UX, and Linux), you must provide both a non-threaded and a threaded version of the API exit module. The threaded version must have an `_r` suffix. The threaded version of the WebSphere MQ application stub implicitly appends `_r` to the given module name before it is loaded.

The length of this field is limited to the maximum path length the platform supports.

Data=data_name
Data to be passed to the API exit in the ExitData field of the MQAXP structure.

If you include this attribute, leading and trailing blanks are removed, the remaining string is truncated to 32 characters, and the result is passed to the exit. If you omit this attribute, the default value of 32 blanks is passed to the exit.

The maximum length of this field is 32 characters.

Sequence=sequence_number
The sequence in which this API exit is called relative to other API exits. An exit with a **low** sequence number is called before an exit with a **higher** sequence number. There is no need for the sequence numbering of exits to be contiguous; a sequence of 1, 2, 3 has the same result as a sequence of 7, 42, 1096. If two exits have the same sequence number, the queue manager decides which one to call first. You can tell which was called *after* the event by putting the time or a marker in ExitChainArea indicated by the ExitChainAreaPtr in MQAXP or by writing your own log file.

This attribute is an unsigned numeric value.

Sample stanzas

The mqs.ini file below contains the following stanzas:

ApiExitTemplate
This stanza defines an exit with the descriptive name `OurPayrollQueueAuditor`, module name `auditor`, and sequence number 2. A data value of 123 is passed to the exit.

ApiExitCommon

This stanza defines an exit with the descriptive name `MQPoliceman`, module name `tmqp`, and sequence number 1. The data passed is an instruction (`CheckEverything`).

`mqs.ini`

```
ApiExitTemplate:
   Name=OurPayrollQueueAuditor
   Sequence=2
   Function=EntryPoint
   Module=/opt/ABC/auditor
   Data=123
ApiExitCommon:
   Name=MQPoliceman
   Sequence=1
   Function=EntryPoint
   Module=/opt/MQPolice/tmqp
   Data=CheckEverything
```

The qm.ini file below contains an ApiExitLocal definition of an exit with the descriptive name `ClientApplicationAPIchecker`, module name `ClientAppChecker`, and sequence number 3.

`qm.ini`

```
ApiExitLocal:
   Name=ClientApplicationAPIchecker
   Sequence=3
   Function=EntryPoint
   Module=/opt/Dev/ClientAppChecker
   Data=9.20.176.20
```

Changing the configuration information

The WebSphere MQ configuration file, mqs.ini, contains information relevant to all the queue managers on a particular node. You can find it in the `/var/mqm` directory.

A queue manager configuration file, qm.ini, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager, held in the root of the directory tree occupied by the queue manager. For example, the path and the name for a configuration file for a queue manager called QMNAME is:

`/var/mqm/qmgrs/QMNAME/qm.ini`

Before editing a configuration file, back it up so that you have a copy you can revert to if the need arises.

You can edit configuration files either:

- Automatically, using commands that change the configuration of queue managers on the node
- Manually, using a standard text editor

If you set an incorrect value on a configuration file attribute, the value is ignored and an operator message is issued to indicate the problem. (The effect is the same as missing out the attribute entirely.)

Configuring API exits on Windows systems

You configure API exits on Windows systems using the WebSphere MQ Services snap-in or the **amqmdain** command to update the Windows Registry.

A new property page for the IBM WebSphere MQ Services node, **API Exits** describes the two types of API exit managed from this node: ApiExitCommon and ApiExitTemplate. In the Exits property page for individual queue managers, you can update the ApiExitLocal. `Configure...` buttons launch a dialog to manage the entries within each stanza. The dialog consists of a multicolumn list of any API exits already defined in the appropriate stanza, with buttons to add, view, and change the properties of exits, and remove them.

When entering, or changing, the attributes for an exit, the attributes are those defined in "Attributes for all stanzas" on page 421.

When you finish defining or changing an exit, press OK to update the Registry. To discard the changes you have made, press Cancel.

Chapter 24. API exit reference information

This chapter provides reference information for the API exit. It includes:

- Data structures used by an API exit function:
  - "MQACH – API exit chain header" on page 427
  - "MQAXC – API exit context" on page 430
  - "MQAXP – API exit parameter" on page 434
- Calls an API exit function can issue:
  - "MQXEP – Register entry point" on page 442
- Definitions of the API exit functions:
  - "MQ_BACK_EXIT – Back out changes" on page 445
  - "MQ_BEGIN_EXIT – Begin unit of work" on page 446
  - "MQ_CLOSE_EXIT – Close object" on page 447
  - "MQ_CMIT_EXIT – Commit changes" on page 448
  - "MQ_CONNX_EXIT – Connect queue manager (extended)" on page 449
  - "MQ_DISC_EXIT – Disconnect queue manager" on page 451
  - "MQ_GET_EXIT – Get message" on page 452
  - "MQ_INIT_EXIT – Initialize exit environment" on page 454
  - "MQ_INQ_EXIT – Inquire object attributes" on page 455
  - "MQ_OPEN_EXIT – Open object" on page 457
  - "MQ_PUT_EXIT – Put message" on page 458
  - "MQ_PUT1_EXIT – Put one message" on page 460
  - "MQ_SET_EXIT – Set object attributes" on page 462
  - "MQ_TERM_EXIT – Terminate exit environment" on page 464

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 MQ calls other than MQXEP.
3. All other exit functions can issue the following MQ calls:

   MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
4. If an exit function issues the MQCONN call, or the MQCONNX call with the MQCNO_HANDLE_SHARE_NONE option, 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. If an exit function issues the MQCONNX call with the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options, the call returns a new shared handle. This provides the exit suite with a connection handle of its own, and hence a unit of work that is independent of the application's unit of work. The exit suite can use this handle to put and get messages within its own unit of work, and commit or back out that unit of work; all of this can be done without affecting the application's unit of work in any way.

Because the exit function is using a connection handle that is different from the handle being used by the application, MQ calls issued by the exit function result in the relevant API exit functions being invoked. Exit functions can therefore be invoked recursively. Note that both the *ExitUserArea* field in MQAXP and the exit chain area have connection-handle scope. Consequently, an exit function cannot use those areas to signal to another instance of itself invoked recursively that it is already active.

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). However, the exit can cause the application to exceed system limits sooner than would otherwise be the case (for example, by exceeding the maximum number of uncommitted messages in a unit of work).

   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). 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

The following table summarizes the fields in the structure.

Table 29. Fields in MQACH

Field	Description	Page
StrucId	Structure identifier	427
Version	Structure version number	428
StrucLength	Length of MQACH structure	428
ChainAreaLength	Total length of chain area	428
ExitInfoName	Exit information name	429
NextChainAreaPtr	Address of next chain area	429

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 `malloc`), 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 the following 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	Page
StrucId	Structure identifier	430
Version	Structure version number	430
Environment	Environment	431
UserId	User identifier	431
SecurityId	Security identifier	431
ConnectionName	Connection name	432
LongMCAUserIdLength	Length of long MCA user identifier	432
LongRemoteUserIdLength	Length of long remote user identifier	432
LongMCAUserIdPtr	Address of long MCA user identifier	432
LongRemoteUserIdPtr	Address of long remote user identifier	432
ApplName	Application name	432
ApplType	Application type	432
ProcessId	Process identifier	433
ThreadId	Thread identifier	433

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 the following 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_MQSC
The "runmqsc" command interpreter.

MQXE_MCA
Message channel agent.

MQXE_MCA_SVRCONN
Message channel agent acting on behalf of a client.

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. This name is obtained in the same way as the default value for the *PutApplName* field in MQMD.

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.

This is the same identifier used in WebSphere MQ trace and FFST dumps, but might be different from the operating system process identifier. Where applicable, the exit handler sets this field on entry to each exit function.

This is an input field to the exit.

ThreadId (MQTID)

The WebSphere MQ thread identifier.

This is the same identifier used in WebSphere MQ trace and FFST dumps, but might be different from the operating system thread identifier. Where applicable, the exit handler sets this field on entry to each exit function.

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

The following table summarizes the fields in the structure.

Table 31. Fields in MQAXP

Field	Description	Page
StrucId	Structure identifier	434
Version	Structure version number	434
ExitId	Type of exit	435
ExitReason	Reason for invoking exit	435
ExitResponse	Response from exit	436
ExitResponse2	Secondary response from exit	437
Feedback	Feedback code	438
APICallerType	API caller type	438
ExitUserArea	Exit user area	438
ExitData	Exit data	439
ExitInfoName	Exit information name	439
ExitPDArea	Problem determination area	439
QMgrName	Name of local queue manager	439
ExitChainAreaPtr	Address of first chain area	439
Hconfig	Configuration handle	440
Function	API function identifier	440

The MQAXP structure describes the information that is passed to an API exit.

Fields

The MQAXP structure contains the following 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_CONNX
: MQCONNX call.

MQXF_DISC
: MQDISC call.

MQXF_OPEN
: MQOPEN call.

MQXF_CLOSE
: MQCLOSE call.

MQXF_PUT1
: MQPUT1 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_BEGIN
MQBEGIN 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 the following 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_CONNX
MQCONNX call.
MQXF_DISC
MQDISC call.
MQXF_OPEN
MQOPEN call.
MQXF_CLOSE
MQCLOSE call.
MQXF_PUT1
MQPUT1 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_BEGIN
MQBEGIN 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. If the unit of work is being coordinated by an external unit-of-work manager, MQ_BACK_EXIT is also invoked in response to the application issuing the unit-of-work manager's back-out call.

Syntax

MQ_BACK_EXIT *(pExitParms, pExitContext, pHconn, pCompCode, pReason)*

Parameters

The MQ_BACK_EXIT call has the following 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_BEGIN_EXIT – Begin unit of work

Exit providers can supply an MQ_BEGIN_EXIT function to intercept the MQBEGIN call.

Syntax

MQ_BEGIN_EXIT *(pExitParms, pExitContext, pHconn, ppBeginOptions, pCompCode, pReason)*

Parameters

The MQ_BEGIN_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output
Exit parameter structure.

pExitContext (PMQAXC) – input/output
Exit context structure.

pHconn (PMQHCONN) – input/output
Connection handle.

ppBeginOptions (PPMQBO) – input/output
Options that control the action of MQBEGIN.

pCompCode (PMQLONG) – input/output
Completion code.

pReason (PMQLONG) – input/output
Reason code qualifying *pCompCode*.

C invocation

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn,
               &pBeginOptions, &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 */
PPMQBO    ppBeginOptions;  /* Options that control the action of
                              MQBEGIN */
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 the following 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. If the unit of work is being coordinated by an external unit-of-work manager, MQ_CMIT_EXIT is also invoked in response to the application issuing the unit-of-work manager's commit call.

Syntax

MQ_CMIT_EXIT *(pExitParms, pExitContext, pHconn, pCompCode, pReason)*

Parameters

The MQ_CMIT_EXIT call has the following 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 the following 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, separate entry points are defined for these two calls. To intercept *both* calls, the MQXEP call must be used at least twice – once with function identifier MQXF_CONN, and again with MQXF_CONNX.

   Because the MQ_CONNX_EXIT interface is the same for MQCONN and MQCONNX, a single exit function can be used for both calls; the *Function* field in the MQAXP structure indicates which call is in progress. Alternatively, the MQXEP call can be used to register different exit functions for the two calls.
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 the following 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 the following 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 the following 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 the following 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:

MQ_INQ_EXIT

```
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 the following 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.

Syntax

```
MQ_PUT_EXIT (pExitParms, pExitContext, pHconn, pHobj, ppMsgDesc,
       ppPutMsgOpts, pBufferLength, ppBuffer, pCompCode, pReason)
```

Parameters

The MQ_PUT_EXIT call has the following 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*.

Usage notes

- Report messages generated by the queue manager skip the normal call processing. As a result, such messages cannot be intercepted by the MQ_PUT_EXIT function or the MQPUT1 function. However, report messages generated by the message channel agent are processed normally, and hence can be intercepted by the MQ_PUT_EXIT function or the MQ_PUT1_EXIT function. To be sure to intercepting all of the report messages generated by the MCA, both MQ_PUT_EXIT and MQ_PUT1_EXIT should be used.

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_PUT1_EXIT – Put one message

Exit providers can supply an MQ_PUT1_EXIT function to intercept the MQPUT1 call.

Syntax

```
MQ_PUT1_EXIT (pExitParms, pExitContext, pHconn, ppObjDesc,
        ppMsgDesc, ppPutMsgOpts, pBufferLength, ppBuffer, pCompCode, pReason)
```

Parameters

The MQ_PUT1_EXIT call has the following 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.

ppMsgDesc (PPMQMD) – input/output
Message descriptor.

ppPutMsgOpts (PPMQPMO) – input/output
Options that control the action of MQPUT1.

pBufferLength (PMQLONG) – input/output
Length of the message in *ppBuffer*.

ppBuffer (PPMQVOID) – input/output
Message data.

pCompCode (PMQLONG) – input/output
Completion code.

pReason (PMQLONG) – input/output
Reason code qualifying *pCompCode*.

C invocation

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn,
              &pObjDesc, &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 */
PPMQOD    ppObjDesc;      /* Object descriptor */
PPMQMD    ppMsgDesc;      /* Message descriptor */
PPMQPMO   ppPutMsgOpts;   /* Options that control the action of MQPUT1 */
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 the following 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 the following 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 */
```

Chapter 25. WebSphere MQ constants

This appendix specifies the values of the named constants that apply to installable services and the API exit.

The constants are grouped according to the parameter or field to which they relate. All of the names of the constants in a group begin with a common prefix of the form "MQ*xxxxx*_", where *xxxxx* represents a string of 0 through 5 characters that indicates the parameter or field to which the values relate. The constants are ordered alphabetically by this prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation `X'hhhh'`, where each "h" denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol "b".

List of constants

The following sections list all the named constants that are mentioned in this book, and show their values.

MQ_* (Lengths of character string and byte fields)

MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_EXIT_DATA_LENGTH	32	X'00000020'
MQ_EXIT_INFO_NAME_LENGTH	48	X'00000030'
MQ_EXIT_PD_AREA_LENGTH	48	X'00000030'
MQ_EXIT_USER_AREA_LENGTH	16	X'00000010'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_SECURITY_ID_LENGTH	40	X'00000028'
MQ_USER_ID_LENGTH	12	X'0000000C'

MQACH_* (API exit chain header length)

MQACH_LENGTH_1	(variable)
MQACH_CURRENT_LENGTH	(variable)

MQACH_* (API exit chain header structure identifier)

MQACH_STRUC_ID	'ACHb'

For the C programming language, the following array version is also defined:

MQACH_STRUC_ID_ARRAY	'A','C','H','b'	

MQACH_* (API exit chain header version)

MQACH_VERSION_1	1	X'00000001'
MQACH_CURRENT_VERSION	1	X'00000001'

MQAXC_* (API exit context structure identifier)

MQAXC_STRUC_ID	'AXCb'

For the C programming language, the following array version is also defined:

MQAXC_STRUC_ID_ARRAY	'A','X','C','b'

MQAXC_* (API exit context version)

MQAXC_VERSION_1	1	X'00000001'
MQAXC_CURRENT_VERSION	1	X'00000001'

MQAXP_* (API exit parameter structure identifier)

MQAXP_STRUC_ID	'AXPb'

For the C programming language, the following array version is also defined:

MQAXP_STRUC_ID_ARRAY	'A','X','P','b'

MQAXP_* (API exit parameter version)

MQAXP_VERSION_1	1	X'00000001'
MQAXP_CURRENT_VERSION	1	X'00000001'

MQCC_* (Completion code)

MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'
MQCC_FAILED	2	X'00000002'

MQFB_* (Feedback)

MQFB_NONE	0	X'00000000'
MQFB_SYSTEM_FIRST	1	X'00000001'
MQFB_SYSTEM_LAST	65535	X'0000FFFF'
MQFB_APPL_FIRST	65536	X'00010000'
MQFB_APPL_LAST	999999999	X'3B9AC9FF'

MQOT_* (Object type)

MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_Q_MGR	5	X'00000005'
MQOT_AUTH_INFO	7	X'00000007'
MQOT_RESERVED_1	999	X'000003E7'

MQRC_* (Reason code)

MQRC_NONE	0	X'00000000'
MQRC_BUFFER_LENGTH_ERROR	2005	X'000007D5'
MQRC_NOT_AUTHORIZED	2035	X'000007F3'
MQRC_RESOURCE_PROBLEM	2102	X'00000836'
MQRC_SUPPRESSED_BY_EXIT	2109	X'0000083D'
MQRC_UNEXPECTED_ERROR	2195	X'00000893'
MQRC_HCONFIG_ERROR	2280	X'000008E8'
MQRC_FUNCTION_ERROR	2281	X'000008E9'
MQRC_SERVICE_NOT_AVAILABLE	2285	X'000008ED'
MQRC_INITIALIZATION_FAILED	2286	X'000008EE'
MQRC_TERMINATION_FAILED	2287	X'000008EF'
MQRC_UNKNOWN_Q_NAME	2288	X'000008F0'
MQRC_SERVICE_ERROR	2289	X'000008F1'
MQRC_Q_ALREADY_EXISTS	2290	X'000008F2'
MQRC_USER_ID_NOT_AVAILABLE	2291	X'000008F3'
MQRC_UNKNOWN_ENTITY	2292	X'000008F4'
MQRC_UNKNOWN_REF_OBJECT	2294	X'000008F6'
MQRC_WRONG_CF_LEVEL	2366	X'0000093E'
MQRC_API_EXIT_ERROR	2374	X'00000946'
MQRC_API_EXIT_INIT_ERROR	2375	X'00000947'
MQRC_API_EXIT_TERM_ERROR	2376	X'00000948'
MQRC_EXIT_REASON_ERROR	2377	X'00000949'
MQRC_RESERVED_VALUE_ERROR	2378	X'0000094A'
MQRC_NO_DATA_AVAILABLE	2379	X'0000094B'

MQSID_* (Security identifier)

MQSID_NONE	X'00...00' (40 nulls)

For the C programming language, the following array version is also defined:

MQSID_NONE_ARRAY	'\0','\0',...'\0','\0'

MQXACT_* (API exit caller type)

MQXACT_EXTERNAL	1	X'00000001'
MQXACT_INTERNAL	2	X'00000002'

MQXCC_* (Exit response)

MQXCC_FAILED	-8	X'FFFFFFF8'
MQXCC_SUPPRESS_EXIT	-5	X'FFFFFFFB'
MQXCC_SKIP_FUNCTION	-2	X'FFFFFFFE'
MQXCC_SUPPRESS_FUNCTION	-1	X'FFFFFFFF'
MQXCC_OK	0	X'00000000'

MQXE_* (API exit environment)

MQXE_OTHER	0	X'00000000'
MQXE_MCA	1	X'00000001'
MQXE_MCA_SVRCONN	2	X'00000002'
MQXE_COMMAND_SERVER	3	X'00000003'
MQXE_MQSC	4	X'00000004'

MQXF_* (API exit function identifier)

MQXF_INIT	1	X'00000001'
MQXF_TERM	2	X'00000002'
MQXF_CONN	3	X'00000003'
MQXF_CONNX	4	X'00000004'
MQXF_DISC	5	X'00000005'
MQXF_OPEN	6	X'00000006'
MQXF_CLOSE	7	X'00000007'
MQXF_PUT1	8	X'00000008'
MQXF_PUT	9	X'00000009'
MQXF_GET	10	X'0000000A'
MQXF_DATA_CONV_ON_GET	11	X'0000000B'
MQXF_INQ	12	X'0000000C'
MQXF_SET	13	X'0000000D'
MQXF_BEGIN	14	X'0000000E'
MQXF_CMIT	15	X'0000000F'
MQXF_BACK	16	X'00000010'

MQXPDA_* (API exit problem determination area)

MQXPDA_NONE	X'00...00' (48 nulls)

For the C programming language, the following array version is also defined:

MQXPDA_NONE_ARRAY	'\0','\0',...'\0','\0'

MQXR_* (Exit reason)

MQXR_BEFORE	1	X'00000001'

MQXR_AFTER	2	X'00000002'
MQXR_CONNECTION	3	X'00000003'

MQXR2_* (Secondary exit response)

MQXR2_DEFAULT_CONTINUATION	0	X'00000000'
MQXR2_CONTINUE_CHAIN	8	X'00000008'
MQXR2_SUPPRESS_CHAIN	16	X'00000010'

MQXT_* (Exit identifier)

MQXT_API_EXIT	2	X'00000002'

MQXUA_* (Exit user area)

MQXUA_NONE	X'00...00' (16 nulls)

For the C programming language, the following array version is also defined:

MQXUA_NONE_ARRAY	'\0','\0',...'\0','\0'

MQZAD_* (Authority data structure identifier)

MQZAD_STRUC_ID	'ZADb'

For the C programming language, the following array version is also defined:

MQZAD_STRUC_ID_ARRAY	'Z','A','D','b'

MQZAD_* (Authority data version)

MQZAD_VERSION_1	1	X'00000001'
MQZAD_CURRENT_VERSION	1	X'00000001'

MQZAET_* (Authority service entity type)

MQZAET_NONE	0	X'00000000'
MQZAET_PRINCIPAL	1	X'00000001'
MQZAET_GROUP	2	X'00000002'

MQZAO_* (Authority service authorization type)

MQZAO_NONE	0	X'00000000'
MQZAO_CONNECT	1	X'00000001'

MQZAO_BROWSE	2	X'00000002'
MQZAO_INPUT	4	X'00000004'
MQZAO_OUTPUT	8	X'00000008'
MQZAO_INQUIRE	16	X'00000010'
MQZAO_SET	32	X'00000020'
MQZAO_PASS_IDENTITY_CONTEXT	64	X'00000040'
MQZAO_PASS_ALL_CONTEXT	128	X'00000080'
MQZAO_SET_IDENTITY_CONTEXT	256	X'00000100'
MQZAO_SET_ALL_CONTEXT	512	X'00000200'
MQZAO_ALTERNATE_USER_AUTHORITY	1024	X'00000400'
MQZAO_ALL_MQI	2047	X'000007FF'
MQZAO_CREATE	65536	X'00010000'
MQZAO_DELETE	131072	X'00020000'
MQZAO_DISPLAY	262144	X'00040000'
MQZAO_CHANGE	524288	X'00080000'
MQZAO_CLEAR	1048576	X'00100000'
MQZAO_AUTHORIZE	8388608	X'00800000'
MQZAO_ALL_ADMIN	10354688	X'009E0000'
MQZAO_ALL	10356735	X'009E07FF'

MQZAS_* (Authority service version)

MQZAS_VERSION_1	1	X'00000001'
MQZAS_VERSION_2	2	X'00000002'
MQZAS_VERSION_3	3	X'00000003'
MQZAS_VERSION_4	4	X'00000004'

MQZCI_* (Continuation indicator)

MQZCI_DEFAULT	0	X'00000000'
MQZCI_CONTINUE	0	X'00000000'
MQZCI_STOP	1	X'00000001'

MQZED_* (Entity descriptor structure identifier)

MQZED_STRUC_ID	'ZEDb'

For the C programming language, the following array version is also defined:

MQZED_STRUC_ID_ARRAY	'Z','E','D','b'

MQZED_* (Entity descriptor version)

MQZED_VERSION_1	1	X'00000001'
MQZED_CURRENT_VERSION	1	X'00000001'

MQZID_* (Function identifier, all services)

MQZID_INIT	0	X'00000000'
MQZID_TERM	1	X'00000001'

MQZID_* (Function identifier, authority service)

MQZID_INIT_AUTHORITY	0	X'00000000'
MQZID_TERM_AUTHORITY	1	X'00000001'
MQZID_CHECK_AUTHORITY	2	X'00000002'
MQZID_COPY_ALL_AUTHORITY	3	X'00000003'
MQZID_DELETE_AUTHORITY	4	X'00000004'
MQZID_SET_AUTHORITY	5	X'00000005'
MQZID_GET_AUTHORITY	6	X'00000006'
MQZID_GET_EXPLICIT_AUTHORITY	7	X'00000007'
MQZID_REFRESH_CACHE	8	X'00000008'
MQZID_ENUMERATE_AUTHORITY_DATA	9	X'00000009'

MQZID_* (Function identifier, name service)

MQZID_INIT_NAME	0	X'00000000'
MQZID_TERM_NAME	1	X'00000001'
MQZID_LOOKUP_NAME	2	X'00000002'
MQZID_INSERT_NAME	3	X'00000003'
MQZID_DELETE_NAME	4	X'00000004'

MQZID_* (Function identifier, userid service)

MQZID_INIT_USERID	0	X'00000000'
MQZID_TERM_USERID	1	X'00000001'
MQZID_FIND_USERID	2	X'00000002'

MQZIO_* (Initialization options)

MQZIO_PRIMARY	0	X'00000000'
MQZIO_SECONDARY	1	X'00000001'

MQZNS_* (Name service version)

MQZNS_VERSION_1	1	X'00000001'

MQZSE_* (Start-enumeration indicator)

MQZSE_START	1	X'00000001'
MQZSE_CONTINUE	0	X'00000000'

MQZTO_* (Termination options)

MQZTO_PRIMARY	0	X'00000000'
MQZTO_SECONDARY	1	X'00000001'

MQZUS_* (Userid service version)

MQZUS_VERSION_1	1	X'00000001'

Part 8. Appendixes

Appendix A. System and default objects

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

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

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

- Table 32 lists the system and default queue objects.
- Table 33 on page 476 lists the system and default channel objects.
- Table 34 on page 476 lists the system and default namelist objects.
- Table 35 on page 476 lists the system and default process objects.

Table 32. System and default objects: queues

Object name	Description
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.MQSC.REPLY.QUEUE	MQSC command reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.PENDING.DATA.QUEUE	Support deferred messages in JMS.

Default objects

Table 33. System and default objects: channels

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

Table 34. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 35. System and default objects: processes

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

Windows default configuration objects

On Windows systems, you set up a default configuration using either the WebSphere MQ First Steps application or the WebSphere MQ Postcard application.

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

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

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

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

Windows default configuration

The cluster name for the Windows default configuration is DEFAULT_CLUSTER.

If the queue manager is not a repository queue manager, the objects listed in Table 36 are created.

Table 36. Objects created by the Windows default configuration application

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

If the queue manager is a repository queue manager, the default configuration is similar to that described in Table 36, but with the following differences:

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

Windows default configuration

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

Appendix B. Directory structure (Windows systems)

Table 37 shows the directories found under the root `c:\Program Files\IBM\WebSphere MQ\`. If you have installed WebSphere MQ for Windows under a different directory, the root is modified appropriately.

Table 37. WebSphere MQ for Windows directory structure

\bin	Binary files (commands and DDLs).
\config	Configuration information.
\conv	Files for data conversion in folder `\table`.
\errors	The operator message files, from newest to oldest: AMQERR01.LOG AMQERR02.LOG AMQERR03.LOG This folder also holds any FFDC files that are produced.
\exits	Channel exit programs.
\licenses	A folder for each national language. Each folder contains license information.
\log	A folder for each queue manager. The following subdirectories and files will exist for each queue manager after you have been using that queue manager for some time. **AMQHLCTL.LFH** Log control file. **Active** This directory contains the log files numbered S0000000.LOG, S0000001.LOG, S00000002.LOG, and so on.
\qmgrs	Folder `\@SYSTEM\errors`, containing error logs for problems not associated with a particular queue manager. Also contains a folder for each queue manager; the contents of this folder are described in Table 38 on page 480.
\tivoli	The signature file used by Tivoli.
\tools	All the WebSphere MQ sample programs. These are described in *WebSphere MQ for Windows, V5.3 Quick Beginnings*.
\uninst	Files necessary to uninstall WebSphere MQ.

Table 38 on page 480 shows the directory structure for each queue manager in the `c:\Program Files\IBM\WebSphere MQ\qmgrs\` folder. The queue manager might have been transformed as described in "Understanding WebSphere MQ file names" on page 18.

Directory structure (Windows systems)

Table 38. Content of a \queue-manager-name\ folder for WebSphere MQ for Windows

amqalchk.fil	Checkpoint file containing information about the last checkpoint.
\@ipcc	Folder containing the channel tables.
\dce	Directory reserved for use by DCE support.
\errors	The operator message files, from newest to oldest: AMQERR01.LOG AMQERR02.LOG AMQERR03.LOG
\namelist	A file for each WebSphere MQ namelist.
\authinfo	A file for each authentication information object.
\Plugcomp	Directory reserved for use by WebSphere MQ installable services.
\Procdef	Each WebSphere MQ process definition has a file in here. Where possible, the file name matches the associated process definition name, but some characters have to be altered. There might be a directory called @MANGLED here containing process definitions with transformed or mangled names.
\Qmanager	The following files: **Qmanager** The queue manager object. **QMQMOBJCAT** The object catalogue containing the list of all WebSphere MQ objects, used internally. **Note:** If you are using a FAT system, this name is transformed and a subdirectory created containing the file with its name transformed. **QAADMIN** File used internally for controlling authorizations.
\Queues	Each queue has a directory here containing a single file called Q. Where possible, the directory name matches the associated queue name but some characters have to be altered. There might be a directory called @MANGLED here containing queues with transformed or mangled names.
\Startprm	Directory containing temporary files used internally.
\ssl	Directory for SSL certificate stores.

Appendix C. Directory structure (UNIX systems)

Figure 32 on page 482 shows the general layout of the data and log directories associated with a specific queue manager. The directories shown apply to the default installation. If you change this, the locations of the files and directories are modified accordingly. For information about the location of the product files, see one of the following:

- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*

In Figure 32 on page 482, the layout is representative of WebSphere MQ after a queue manager has been in use for some time. The actual structure that you have depends on which operations have occurred on the queue manager.

Directory structure (UNIX systems)

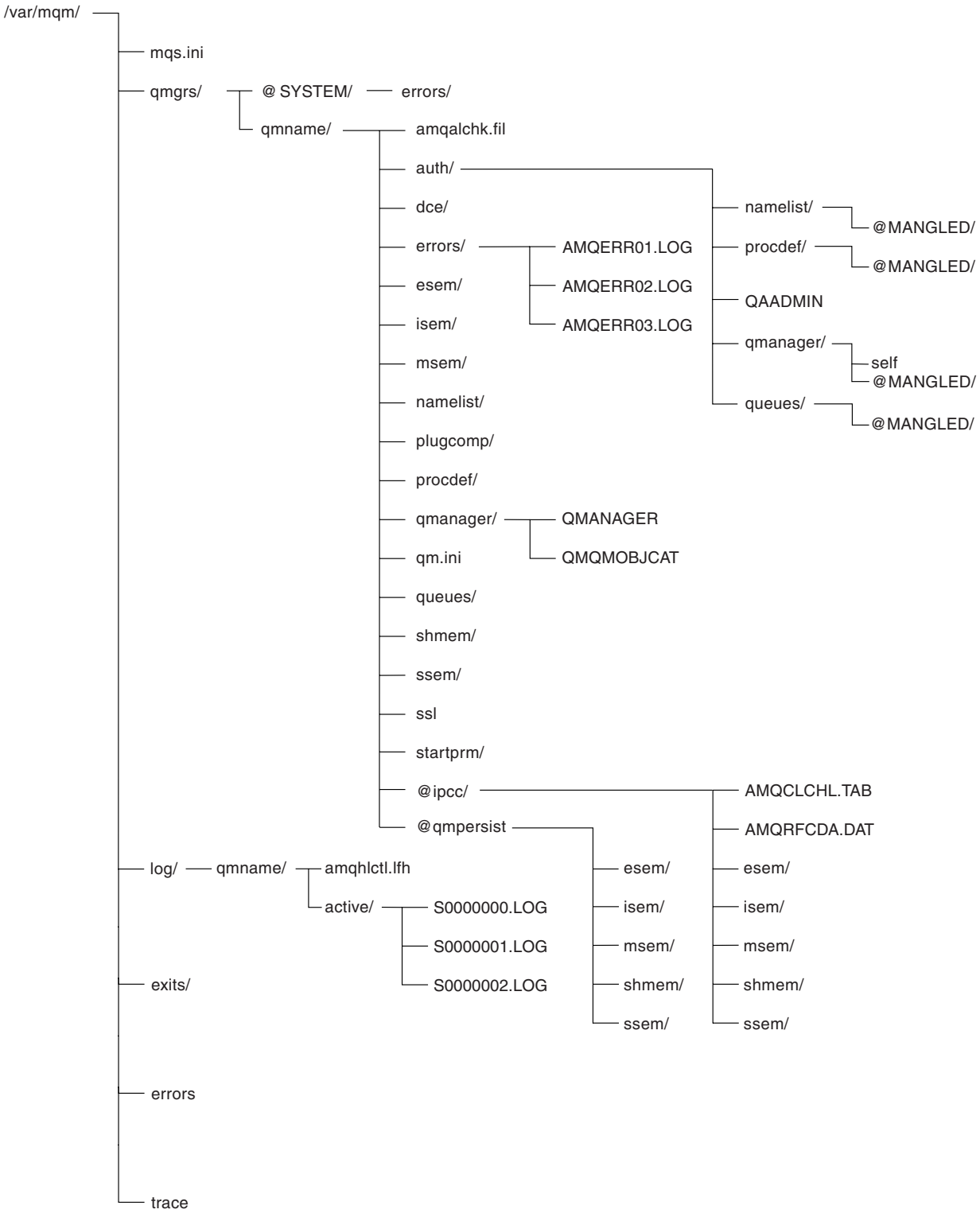


Figure 32. Default directory structure (UNIX systems) after a queue manager has been started

By default, the following directories and files are located in the directory `/var/mqm/qmgrs/qmname/` (where *qmname* is the name of the queue manager).

Table 39. Default content of a /var/mqm/qmgrs/qmname/ directory on UNIX systems

amqalchk.fil	Checkpoint file containing information about the last checkpoint.
auth/	Subdirectories and files associated with authority. **@MANGLED** This file contains the authority stanzas for all classes. **namelist/** This directory contains a file for each namelist. Each file contains the authority stanzas for the associated namelist. **@MANGLED** This file contains the authority stanzas for the namelist. **procdef/** This directory contains a file for each process definition. Each file contains the authority stanzas for the associated process definition. **@MANGLED** This file contains the authority stanzas for the process definition class. **QAADMIN** File used internally for controlling authorizations. **qmanager/** **@MANGLED** This file contains the authority stanzas for the queue manager class. **self** This file contains the authority stanzas for the queue manager object. **queues/** This directory contains a file for each queue. Each file contains the authority stanzas for the associated queue. **@MANGLED** This file contains the authority stanzas for the queue class.
dce/	Empty directory reserved for use by DCE support.
errors/	Directory containing FFSTs, client application errors, and operator message files from newest to oldest: AMQERR01.LOG AMQERR02.LOG AMQERR03.LOG
esem/	Directory containing files used internally.
isem/	Directory containing files used internally.
msem/	Directory containing files used internally.
namelist/	Each WebSphere MQ namelist definition is associated with a file in this directory. The file name matches the namelist definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 18.
plugcomp/	Empty directory reserved for use by installable services.

Table 39. Default content of a /var/mqm/qmgrs/qmname/ directory on UNIX systems (continued)

procdef/	Each WebSphere MQ process definition is associated with a file in this directory. The file name matches the process definition name—subject to certain restrictions; see "Understanding WebSphere MQ file names" on page 18.
qmanager/	**QMANAGER** The queue manager object. **QMQMOBJCAT** The object catalog containing the list of all WebSphere MQ objects; used internally.
qm.ini	Queue manager configuration file.
queues/	Each queue has a directory in here containing a single file called q. The file name matches the queue name, subject to certain restrictions; see "Understanding WebSphere MQ file names" on page 18.
shmem/	Directory containing files used internally.
ssem/	Directory containing files used internally.
ssl	Directory for SSL key database files.
startprm/	Directory containing temporary files used internally.
@ipcc/	**AMQCLCHL.TAB** Client channel table file. **AMQRFCDA.DAT** Channel table file. **esem/** Directory containing files used internally. **isem/** Directory containing files used internally. **msem/** Directory containing files used internally. **shmem/** Directory containing files used internally. **ssem/** Directory containing files used internally.
@qmpersist	**esem/** Directory containing files used internally. **isem/** Directory containing files used internally. **msem/** Directory containing files used internally. **shmem/** Directory containing files used internally. **ssem/** Directory containing files used internally.

By default, the following directories and files are found in /var/mqm/log/*qmname*/ (where *qmname* is the name of the queue manager).

The following subdirectories and files exist after you have installed WebSphere MQ, created and started a queue manager, and have been using that queue manager for some time.

amqhlctl.lfh	Log control file.
active/	This directory contains the log files numbered S0000000.LOG, S0000001.LOG, S0000002.LOG, and so on.

Appendix D. Stopping and removing queue managers manually

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

Stopping a queue manager manually

The standard way of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, you can use one of the procedures described here to stop it manually.

Stopping queue managers in WebSphere MQ for Windows

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

1. List the names (IDs) of the processes currently running using the Windows Process Viewer (PView)
2. Stop the processes using PView in the following order (if they are running):

AMQPCSEA.EXE	The command server
AMQHASMN.EXE	The logger
AMQHARMN.EXE	Log formatter (linear logs only)
AMQZLLP0.EXE	Checkpoint process
AMQZLAA0.EXE	LQM agents
AMQZFUMA.EXE	OAM process
AMQZTRCN.EXE	Trace
AMQZXMA0.EXE	Execution controller
AMQXSSVN.EXE	Shared memory servers
AMQRRMFA.EXE	The repository process (for clusters)
AMQZDMAA	Deferred message processor
AMQRMPPA	Channel receiver

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

Stopping queue managers in WebSphere MQ for UNIX systems

To stop a queue manager running under WebSphere MQ for UNIX systems:

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

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

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

   End the processes in the following order:

amqpcsea	Command server
amqhasmx	Logger

amqharmx	Log formatter (linear logs only)
amqzllp0	Checkpoint processor
amqzlaa0	Queue manager agents
amqzfuma	OAM process
amqzxma0	Processing controller
amqrrmfa	Repository process (for clusters)
amqzdmaa	Deferred message processor
amqrmppa	Channel receiver

Note: Processes that fail to stop can be ended using **kill -9**.

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

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

Attention!
If you do not shut down a queue manager properly, you run the risk of WebSphere MQ not tidying up operating system resources such as semaphores and shared memory sets. This can result in a gradual degradation of system performance and in you having to reboot your system.

Removing queue managers manually

If you want to delete the queue manager after stopping it manually, use the **dltmqm** command. If, for some reason, this command fails to delete the queue manager, use the manual processes described here.

Removing queue managers in WebSphere MQ for Windows

If you encounter problems with the **dltmqm** command in WebSphere MQ for Windows, use the following procedure to delete a queue manager:

1. Type `REGEDIT` from the command prompt to start the Registry Editor.
2. Select the HKEY_LOCAL_MACHINE window.
3. Navigate the tree structure in the left-hand pane of the Registry Editor to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion
   ```

   Make a note of the values within this key called WorkPath and LogPath. Within each of the directories named by these values, you are going to delete a subdirectory containing the data for the queue manager that you are trying to delete. You now need to find out the name of the subdirectory which corresponds to your queue manager.
4. Navigate the tree structure to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
   Configuration\QueueManager
   ```

   Within this key there is a key for each of the queue managers on this computer containing the configuration information for the queue manager. The name of this queue manager key is the name of the subdirectory in which the queue manager's data is stored in the file system. By default, this name is

the same as the queue manager name, but the name might be a transformation of the queue manager name.

5. Examine the keys within the current key. Look for the key that contains a value called `Name`. `Name` contains the name of the queue manager you are trying to delete. Make a note of the name of the key containing the name of the queue manager you are trying to delete. This is the subdirectory name.
6. Locate the queue manager data directory. The name of this directory is the WorkPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
7. Locate the queue manager's log directory. The name of this directory is the LogPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
8. Remove the registry entries that refer to the deleted queue manager. First, navigate the tree structure in the Registry Editor to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
   Configuration\DefaultQueueManager
   ```
9. If the value called Name within this key matches the name of the queue manager you are deleting, delete the DefaultQueueManager key.
10. Navigate the tree to the following key:

    ```
    HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
    Configuration\Services
    ```
11. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.
12. Navigate the tree to the following key:

    ```
    HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
    Configuration\QueueManager
    ```
13. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.

Removing queue managers from the automatic startup list

If for any reason the WebSphere MQ Services snap-in cannot be used to change the startup state of a particular queue manager, use the following routine to carry out the same procedure manually:

1. Stop the WebSphere MQ Services snap-in either from the task bar icon or from the control panel.
2. Type `REGEDIT` on the command line.
3. Select the HKEY_LOCAL_MACHINE window.
4. Navigate the tree structure to find the following key:

   ```
   LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\Configuration\
   Services\<QMgrName>\QueueManager
   ```
5. Change the startup value to zero. (1 means automatic and 0 means manual.)
6. Close the Registry Editor.
7. Run **amqdain regsec**.

Removing queue managers in WebSphere MQ for UNIX systems

The manual removal of a queue manager is potentially very disruptive, particularly if multiple queue managers are being used on a single system. This is because, to completely remove a queue manager, you must delete files, shared

memory, and semaphores. As it is impossible to identify which shared memory and semaphores belong to a particular queue manager, it is necessary to stop all running queue managers.

If you need to delete a queue manager manually, use the following procedure:

1. Stop all queue managers running on the machine from which you need to remove the queue manager.
2. Locate the queue manager directory from the configuration file `/var/mqm/mqs.ini`. To do this, look for the `QueueManager` stanza naming the queue manager to be deleted.

   Its Prefix and Directory attributes identify the queue manager directory. For a Prefix attribute of <Prefix> and a Directory attribute of <Directory>, the full path to the queue manager directory is: `<Prefix>/qmgrs/<Directory>`
3. Locate the queue manager log directory from the qm.ini configuration file in the queue manager directory. The LogPath attribute of the `Log` stanza identifies this directory.
4. Delete the queue manager directory, all subdirectories and files.
5. Delete the queue manager log directory, all subdirectories and files.
6. Remove the queue manager's `QueueManager` stanza from the `/var/mqm/mqs.ini` configuration file.
7. If the queue manager being deleted is also the default queue manager, remove the `DefaultQueueManager` stanza from the `/var/mqm/mqs.ini` configuration file.
8. Remove all shared memory and semaphores owned by the mqm user ID and mqm group, or use the **amqiclen** command and pipe in the mqs.ini file, or restart the machine. Shared resources can be identified using the **ipcs** command, and can be removed with the **ipcrm** command.

Appendix E. Comparing command sets

The tables in this appendix compare the facilities available from the different administration command sets, and state whether you can perform each function from within the WebSphere MQ Explorer snap-in and the WebSphere MQ Services snap-in.

Commands for queue manager administration

Table 40. Commands for queue manager administration

PCF commands	MQSC commands	Control commands	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Change Queue Manager	ALTER QMGR	No equivalent	Yes	No
(Create queue manager) [1]	No equivalent	**crtmqm**	Yes	Yes
(Delete queue manager) [1]	No equivalent	**dltmqm**	Yes	Yes
Inquire Queue Manager	DISPLAY QMGR	**dspmq**	Yes	No
(Stop queue manager) [1]	No equivalent	**endmqm**	Yes	Yes
Ping Queue Manager	PING QMGR	No equivalent	No	No
(Start queue manager) [1]	No equivalent	**strmqm**	Yes	Yes
Notes: 1. Not available as PCF commands				

Commands for command server administration

Table 41. Commands for command server administration

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Display command server	No equivalent	No equivalent	**dspmqcsv**	No	Yes
Start command server	No equivalent	No equivalent	**strmqcsv**	No	Yes
Stop command server	No equivalent	No equivalent	**endmqcsv**	No	Yes

Commands for queue administration

Table 42. Commands for queue administration

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE	No equivalent	Yes	No
Clear Queue	CLEAR QUEUE	No equivalent	Yes	No
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	No equivalent	No	No
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE	No equivalent	Yes	No
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE	No equivalent	Yes	No
Inquire Queue	DISPLAY QUEUE	No equivalent	Yes	No
Inquire Queue Names	DISPLAY QUEUE	No equivalent	Yes	No

Commands for process administration

Table 43. Commands for process administration

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Change Process	ALTER PROCESS	No equivalent	Yes	No
Copy Process	DEFINE PROCESS(x) LIKE(y)	No equivalent	No	No
Create Process	DEFINE PROCESS	No equivalent	Yes	No
Delete Process	DELETE PROCESS	No equivalent	Yes	No
Inquire Process	DISPLAY PROCESS	No equivalent	Yes	No
Inquire Process Names	DISPLAY PROCESS	No equivalent	Yes	No

Commands for channel administration

Table 44. Commands for channel administration

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Change Channel	ALTER CHANNEL	No equivalent	Yes	No
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	No equivalent	No	No

Table 44. Commands for channel administration (continued)

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Create Channel	DEFINE CHANNEL	No equivalent	Yes	No
Delete Channel	DELETE CHANNEL	No equivalent	Yes	No
End Listener	No equivalent	**endmqlsr**	No	Yes
Inquire Channel	DISPLAY CHANNEL	No equivalent	Yes	No
Inquire Channel Names	DISPLAY CHANNEL	No equivalent	Yes	No
Ping Channel	PING CHANNEL	No equivalent	Yes	No
Reset Channel	RESET CHANNEL	No equivalent	Yes	No
Resolve Channel	RESOLVE CHANNEL	No equivalent	Yes	No
Start Channel	START CHANNEL	**runmqchl**	Yes	Yes
Start Channel Initiator	START CHINIT	**runmqchi**	No	Yes
Start Channel Listener	START LISTENER	**runmqlsr**	No	Yes
Stop Channel	STOP CHANNEL	No equivalent	Yes	Yes

Other control commands

Table 45. Other control commands

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
WebSphere MQ Services control	No equivalent	No equivalent	**amqmdain**[1]	No	Yes
Create conversion exit	No equivalent	No equivalent	**crtmqcvx**	No	No
Dump log	No equivalent	No equivalent	**dmpmqlog**	No	No
Display authority	No equivalent	No equivalent	**dspmqaut**	No	No
Display files used by objects	No equivalent	No equivalent	**dspmqfls**	No	No
Display formatted trace	No equivalent	No equivalent	**dspmqtrc** [2]	No	No
Display transactions	No equivalent	No equivalent	**dspmqtrn**	No	No
End trace	No equivalent	No equivalent	**endmqtrc** [1]	No	Yes
Record media image	No equivalent	No equivalent	**rcdmqimg**	No	No
Recreate media object	No equivalent	No equivalent	**rcrmqobj**	No	No
Resolve transactions	No equivalent	No equivalent	**rsvmqtrn**	No	No
Run dead-letter queue handler	No equivalent	No equivalent	**runmqdlq**	No	No

Table 45. Other control commands (continued)

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?	WebSphere MQ Services snap-in equivalent?
Run MQSC commands	No equivalent	No equivalent	**runmqsc**	No	No
Run trigger monitor	No equivalent	No equivalent	**runmqtrm**	No	Yes
Run client trigger monitor	No equivalent	No equivalent	**runmqtmc**	No	No
Set or reset authority	No equivalent	No equivalent	**setmqaut**	No	No
Set service connection points	No equivalent	No equivalent	**setmqscp**[3]	No	No
Start WebSphere MQ trace	No equivalent	No equivalent	**strmqtrc** [1]	No	Yes

Notes:

1. Not supported by WebSphere MQ for AIX.
2. Supported by WebSphere MQ for HP-UX, WebSphere MQ for Solaris, and WebSphere MQ for Linux for Intel and Linux for zSeries only.
3. Supported by WebSphere MQ for Windows only.

Appendix F. Using the User Datagram Protocol

WebSphere MQ for AIX supports the User Datagram Protocol (UDP), a part of the Internet suite of protocols, as an alternative to TCP. You might decide to use UDP instead of TCP for your mobile radio network, where you need to reduce the traffic, and therefore the cost, on a packet radio data network.

UDP uses the Internet Protocol (IP) to deliver datagrams, the basic unit of information for UDP.

You can use UDP to send message data between WebSphere MQ for Windows systems and WebSphere MQ for AIX server systems.

Note: UDP is supplied as part of the operating system or TCP/IP suite you are using; you do not need to buy and install a separate UDP product.

This appendix describes how to:
- Configure WebSphere MQ to use UDP, using MQSC commands
- Write an exit program to monitor when data can be sent

It also provides some hints and tips on what to do to tailor the UDP support in WebSphere MQ to suit your needs.

Configuring WebSphere MQ for UDP

You configure WebSphere MQ to use UDP using MQSC commands.

Make sure there is a listener started for UDP by issuing the following command:

```
runmqlsr -m QMgrName -t UDP
```

Notes:

1. You cannot start a UDP listener on AIX using the MQSC command START LISTENER.
2. Using the **runmqlsr** command means that you must **not** add entries in the `/etc/services` and `/etc/inetd.conf` files for UDP on WebSphere MQ for AIX.

Examples of MQSC command files

The following figures show MQSC command files, supplied with WebSphere MQ, with the channel transport type (the TRPTYPE parameter) set to UDP:
- Figure 33 on page 494 shows the file EARTH.TST
- Figure 34 on page 495 shows the file MOON.TST

Both these files are supplied in directory `\mqw\samples` on WebSphere MQ for Windows. They define the queues and channels you use if you follow the procedure for setting up and verifying two queue managers.

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager.
DEFINE QLOCAL('SAMPLE.EARTH.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)

* Define the remote queue.
* The sample application should put messages on this queue.
DEFINE QREMOTE('SAMPLE.EARTH.REMOTE') REPLACE  +
       DESCR('Remote queue defined on EARTH') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.MOON.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('MOON') +
* This is the name local transmission queue to be used
       XMITQ('SAMPLE.EARTH.XMIT')

* Define the channel that will remove messages from the transmission
* queue SAMPLE.EARTH.XMIT and send them to the machine specified
* by CONNAME.
*
* Change CONNAME to the IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('EARTH.TO.MOON')  CHLTYPE(SDR) TRPTYPE(UDP) +
       XMITQ('SAMPLE.EARTH.XMIT') +
       CONNAME('MOON IP machine name') +
       DESCR('Sender channel for messages to queue manager MOON') +
       REPLACE

* Define the channel that will accept messages from the remote
* queue manager on the machine specified by CONNAME.
*
* Change CONNAME to the IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('MOON.TO.EARTH') CHLTYPE(RQSTR) TRPTYPE(UDP) +
       CONNAME('MOON IP machine name') +
       DESCR('Requester channel for messages from queue manager MOON') +
       REPLACE

* Define the local queue where the remote machine will place
* its messages.
* The sample application should get messages from this queue.
DEFINE QLOCAL('SAMPLE.EARTH.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 33. The supplied file EARTH.TST

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager
DEFINE QLOCAL('SAMPLE.MOON.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)


* Define the remote queue.
* The sample application should put messages on this queue
DEFINE QREMOTE('SAMPLE.MOON.REMOTE') REPLACE  +
       DESCR('Remote queue defined on MOON') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.EARTH.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('EARTH') +
* This is the name local transmission queue to be used
       XMITQ('SAMPLE.MOON.XMIT')


* Define the channel that will remove messages from the transmission
* queue SAMPLE.MOON.XMIT and send them to the remote queue
* manager.
DEFINE CHANNEL ('MOON.TO.EARTH') CHLTYPE(SVR) TRPTYPE(UDP) +
       XMITQ('SAMPLE.MOON.XMIT') +
       DESCR('Server channel for messages to queue manager EARTH') +
       REPLACE


* Define the channel that will accept messages from the remote
* queue manager.
DEFINE CHANNEL ('EARTH.TO.MOON')  CHLTYPE(RCVR) TRPTYPE(UDP) +
       DESCR('Receiver channel for messages from queue manager EARTH') +
       REPLACE


* Define the local queue where the remote machine will place
* its messages.
* The sample application should get messages from this queue
DEFINE QLOCAL('SAMPLE.MOON.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 34. The supplied file MOON.TST

The retry exit

WebSphere MQ allows you to write a C language retry exit. The exit allows your application to suspend data being sent on a channel when communication is not possible (for example, when the mobile user is traveling through a tunnel or is temporarily out of range of a transmitter).

The retry exit can be associated with a monitor program that can assess whether the IP connection is available for sending data. The exit has to be built into an AIX library (in the same way as any other WebSphere MQ library).

The exit is normally called before a datagram is about to be sent but is also called to provide other useful signals.

The retry exit is called under five different conditions:

- When the WebSphere MQ channel is first initialized; the ExitReason variable is set to a value of MQXR_INIT.
- When the WebSphere MQ channel is shut down; the ExitReason variable is set to a value of MQXR_TERM.
- Before each datagram is sent; the ExitReason variable is set to a value of MQXR_RETRY.
- When the end of a batch of messages occurs; the ExitReason variable is set to a value of MQXR_END_BATCH.

  The UDP transport layer knows nothing about the end of batches because this is a concept known only at the queue manager level. At this point, however, the transport layer moves from a series of ccxSend() verbs to a single ccxReceive() verb and back again. This change of mode, from ccxSend() to ccxReceive(), is detected and the transport exit is called accordingly.
- When an *information* datagram is received from the remote end of the link; the ExitReason variable is set to a value of MQXR_ACK_RECEIVED.

The following table provides an explanation of the variables.

Variable	Explanation
ExitReason	Reason for invoking the exit (for example MQXR_RETRY).
ExitUserArea	Exit user area. When the exit is first invoked, the user can return a value here. This value is presented in this field for all subsequent invocations of the exit for that channel.
TransportType	Transport type. This always has a value of MQXPT_UDP.
RetryCount	The number of times the data has been retried (zero on first entry to exit).
DataLength	Length of data to be sent (in bytes).
SessionId	Session identifier. This is unique for each channel.
GroupId	Group identifier. Identifies the batch of datagrams currently being sent.
DataId	Data identifier. This is an identifier for each datagram.
ExitResponse	Response from exit. The user fills this in on return with a value (for example, MQXCC_OK).
Feedback	Reserved.

If you want to postpone sending a datagram in response to an ExitReason of MQXR_RETRY, block returning from the exit until it is safe to send the datagram. In all other cases, the return from the exit is immediate.

There are three possible return codes that can be set when returning from the exit:

- MQXCC_OK; the normal response.
- MQXCC_CLOSE_CHANNEL; in response to an ExitReason of MQXR_RETRY, this closes the channel.
- MQXCC_REQUEST_ACK; in response to a ExitReason of MQXR_RETRY, this modifies the datagram about to be sent so that it requests the remote end of the link to send an *information* datagram back to indicate that the node can be reached. If this datagram arrives, the exit is invoked again with an ExitReason of MQXR_ACK_RECEIVED.

Note: If the datagram fails to arrive at the remote node, for any reason, you have to repeat the request on the next datagram sent.

Other information is available to you when the exit is called (see the file CMQXC.H for full details). An example called REXIT is supplied (see various files called REXIT.*).

You can define the retry exit name, and change the name of the library that contains the exit. The library resides in the same a directory as other WebSphere MQ exits. See *WebSphere MQ Intercommunication* for general information about WebSphere MQ exits.

You configure the retry exit by editing the qm.ini file.

Hints and tips

You might need to do one or both of the following to tailor the UDP support in WebSphere MQ to suit your needs:

- Edit the qm.ini configuration file.

  A sample qm.ini configuration file is shipped in the `mqw\qmgrs\` directory on WebSphere MQ for Windows. To use it, copy it to the subdirectory for the queue manager and edit it as required.

  The parameters and values in the sample configuration file are:

```
UDP:
ACKREQ_TIMEOUT = 5
ACKREQ_RETRY = 60
CONNECT_TIMEOUT = 5
CONNECT_RETRY = 60
ACCEPT_TIMEOUT = 5
ACCEPT_RETRY = 60
DROP_PACKETS = 0
BUNCH_SIZE = 8
PACKET_SIZE = 2048
PSEUDO_ACK  = YES
TRANSPORT:
RETRY_EXIT = exitname
```

  See “User datagram protocol (UDP)” on page 110 and “The TRANSPORT stanza” on page 111 for descriptions of each of these attributes and how to code them.

- Keep messages to less than 4 MB.

Appendix G. WebSphere MQ and UNIX System V IPC resources

This information applies to WebSphere MQ running on UNIX systems only.

WebSphere MQ uses System V interprocess communication (IPC) resources (*semaphores* and *shared memory segments*) to store and pass data between system components. These resource are used by queue manager processes and applications that connect to the queue manager. WebSphere MQ clients do not use System V IPC resources. Use the UNIX command **ipcs** to view the number and size of these resources that the system uses.

Clearing WebSphere MQ shared memory resources

When a WebSphere MQ queue manager stops, it releases the IPC resources that it was using back to the system. There are two situations in which this might not happen automatically:

- If applications are connected to the queue manager when it stops (perhaps because the queue manager was shut down using **endmqm -i** or **endmqm -p**), the IPC resources used by these applications might not be released.
- If the queue manager ends abnormally (for example, if an operator issues the system **kill** command), some IPC resources might be left allocated after all queue manager processes have terminated.

In these cases, the IPC resources are not release back to the system until you manually remove them, or restart (**strmqm**) or delete (**dltmqm**) the queue manager.

WebSphere MQ provides a utility to release System V IPC resources allocated by the queue manager but not freed, for one of the above reasons, back to the system.

To check and free any unused System V IPC resources, type (as an authorized user, mqm or root):

On Solaris, HP, Linux

```
/opt/mqm/bin/amqiclen
```

On AIX

```
/usr/mqm/lib/amqiclen
```

This command does not report any status. However, if some WebSphere MQ-allocated resources could not be freed because they were still in use, the return code is nonzero.

Shared memory on AIX and using EXTSHM

By default, AIX uses System V shared memory in a way that is different from other UNIX platforms. This results in a limit of 10 shared memory segments being attached by a single process.

Under WebSphere MQ Version 5.3, queue managers use the AIX extension EXTSHM, which allows more than 10 segments to be attached by a single process. This helps AIX behave more similarly to other UNIX platforms.

To take full advantage of this facility, we strongly recommend that you set the environment variable on (EXTSHM=ON) in the environment of all WebSphere MQ applications before they start. All WebSphere MQ queue manager processes set this variable for the lifetime of their process if it was not set when the queue manager started.

Note: You must set the environment variable before starting the program, because this environment variable is examined as the program is loaded into memory

If a user's WebSphere MQ application chooses not to set this variable, it can still connect and communicate with WebSphere MQ correctly. However, if it tries to use more shared memory than is available in the 10 slots provided for attaching shared memory segments, that request might fail.

The kinds of situations that can increase the number of segments that WebSphere MQ tries to attach are:

- Many threads all attaching to WebSphere MQ
- Large messages transferring between the application and WebSphere MQ

If the WebSphere MQ application itself uses shared memory (or possibly other libraries that do, such as database connections), this can reduce the number of memory slots available for WebSphere MQ to use.

Note: Not all applications support the use of EXTSHM=ON. Do not set this value globally (in `/etc/environment` for example). It is better to set the value locally in the profile of any user who wants to run WebSphere MQ applications.

Appendix H. Example of a log file

This section contains an example of the output from a **dmpmqlog** command. The dump, which started at the LSN of a specific log record, was produced using the following command:

```
dmpmqlog -mtestqm -s0:0:0:44162
```

```
AMQ7701: DMPMQLOG command is starting.
LOG FILE HEADER
***************

counter1  . . . : 23                  counter2  . . . : 23
FormatVersion . : 2                   logtype . . . . : 10
logactive . . . : 3                   loginactive . . : 2
logsize . . . . : 1024       pages
baselsn . . . . : <0:0:0:0>
nextlsn . . . . : <0:0:0:60864>
lowtranlsn  . . : <0:0:0:0>
minbufflsn  . . : <0:0:0:58120>
headlsn . . . . : <0:0:0:58120>
taillsn . . . . : <0:0:0:60863>
logfilepath . . : ""
hflag1  . . . . : 1
               -> CONSISTENT
               -> CIRCULAR
HeadExtentID  . : 1                   LastEID . . . . : 846249092
LogId . . . . . : 846249061           LastCommit  . . : 0
FirstArchNum  . : 4294967295          LastArchNum . . : 4294967295
nextArcFile . . : 4294967295          firstRecFile  . : 4294967295
firstDlteFile . : 4294967295          lastDeleteFile  : 4294967295
RecHeadFile . . : 4294967295          FileCount . . . : 3
frec_trunclsn . : <0:0:0:0>
frec_readlsn  . : <0:0:0:0>
frec_extnum . . : 0                   LastCId . . . . : 0
onlineBkupEnd . : 0                   softmax . . . . : 4194304


LOG RECORD - LSN <0:0:0:44162>
**********

HLG Header: lrecsize 212, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Start Checkpoint (1025)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 192                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

No data for Start Checkpoint Record
```

Figure 35. Example dmpmqlog output (Part 1 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:44374>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Transaction Table (773)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
TranCount . . . : 0


LOG RECORD - LSN <0:0:0:44594>
**********

HLG Header: lrecsize 1836, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Participants (1537)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 1816                LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPH
Version . . . . : 1                   Flags . . . . . : 3
Count . . . . . : 2

Participant Entry 0
RMName  . . . . : DB2 MQBankDB
RMID  . . . . . : 1
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :

Participant Entry 1
RMName  . . . . : DB2 MQBankDB
RMID  . . . . . : 2
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :
```

Figure 35. Example dmpmqlog output (Part 2 of 12)

```
LOG RECORD - LSN <0:0:0:46448>
**********

HLG Header: lrecsize 236, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM End Checkpoint (1026)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 216                  LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

ChkPtLSN  . . . : <0:0:0:44162>
OldestLSN . . . : <0:0:0:0>
MediaLSN  . . . : <0:0:0:0>


LOG RECORD - LSN <0:0:0:52262>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 200                  LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000
```

Figure 35. Example dmpmqlog output (Part 3 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:52482>
**********

HLG Header: lrecsize 730, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 710                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : Queue1
Qid . . . . . . : {Hash 196836031, Counter: 0}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:52262>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                Data.Length . . : 486
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0    ...............Á
00032:  00 00 00 00 00 00 00 01 00 00 00 22 00 00 00 00    ...........".....
00048:  00 00 00 00 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3€-□........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 22 00 00 00 00 00 00 00 00    .......".........
00144:  00 00 00 00 00 00 00 C9 2C B5 C0 25 FF FF FF FF    .......□,µÀ%....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74                test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C                sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 32 31 35 32 30 20 20 20 20 00 00 00 00    10421520    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  50 65 72 73 69 73 74 65 6E 74 20 6D 65 73 73 61    Persistent messa
00464:  67 65 20 70 75 74 20 75 6E 64 65 72 20 73 79 6E    ge put under syn
00480:  63 70 6F 69 6E 74                                  cpoint
```

Figure 35. Example dmpmqlog output (Part 4 of 12)

```
LOG RECORD - LSN <0:0:0:53458>
**********

HLG Header: lrecsize 734, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 714                  LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: NULL
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                   PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                 Data.Length . . : 490
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0    ...............À
00032:  00 00 00 00 00 00 00 01 00 00 00 26 00 00 00 00    ...........&;...
00048:  00 00 00 00 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3€-□........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 26 00 00 00 00 00 00 00 00    .......&;.......
00144:  00 00 00 00 00 00 00 C9 2C B6 D8 DD FF FF FF FF    .......□,.θ.....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74                test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C                sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 33 32 37 30 36 20 20 20 20 00 00 00 00    10432706    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  50 65 72 73 69 73 74 65 6E 74 20 6D 65 73 73 61    Persistent messa
00464:  67 65 20 6E 6F 74 20 70 75 74 20 75 6E 64 65 72    ge not put under
00480:  20 73 79 6E 63 70 6F 69 6E 74                       syncpoint
```

Figure 35. Example dmpmqlog output (Part 5 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:54192>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:52482>

Version . . . . : 1


LOG RECORD - LSN <0:0:0:54408>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000


LOG RECORD - LSN <0:0:0:54628>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Get Message (259)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 220                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : Queue1
Qid . . . . . . : {Hash 196836031, Counter: 0}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:54408>

Version . . . . : 2
SpcIndex  . . . : 1                   QPriority . . . : 0
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 4294967295, Low 4294967295}
```

Figure 35. Example dmpmqlog output (Part 6 of 12)

```
LOG RECORD - LSN <0:0:0:54868>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Get Message (259)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 220                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: NULL
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 2
SpcIndex  . . . : 1                   QPriority . . . : 0
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 4294967295, Low 4294967295}


LOG RECORD - LSN <0:0:0:55108>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:54628>

Version . . . . : 1


LOG RECORD - LSN <0:0:0:55324>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB40000102374657374716D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000
```

Figure 35. Example dmpmqlog output (Part 7 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:55544>
**********

HLG Header: lrecsize 738, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 718                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:55324>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                Data.Length . . : 494
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0    ...............Ã
00032:  00 00 00 00 00 00 00 01 00 00 00 2A 00 00 00 00    ...........*....
00048:  00 00 00 01 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3€-□........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 2A 00 00 00 00 00 00 00 00    .......*........
00144:  00 00 00 00 00 00 00 C9 2C B8 3E E8 FF FF FF FF    .......□,⊕>.....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74               test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C               sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 34 35 38 37 32 20 20 20 20 00 00 00 00    10445872    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  41 6E 6F 74 68 65 72 20 70 65 72 73 69 73 74 65    Another persiste
00464:  6E 74 20 6D 65 73 73 61 67 65 20 70 75 74 20 75    nt message put u
00480:  6E 64 65 72 20 73 79 6E 63 70 6F 69 6E 74          nder syncpoint
```

Figure 35. Example dmpmqlog output (Part 8 of 12)

```
LOG RECORD - LSN <0:0:0:56282>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Prepare Transaction (770)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:55544>

Version . . . . : 1


LOG RECORD - LSN <0:0:0:56498>
**********

HLG Header: lrecsize 708, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Prepared (1538)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 688                 LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPR
Version . . . . : 1                   Flags . . . . . : 1
Count . . . . . : 3

Participant Entry 0
RMID  . . . . . : 0                   State . . . . . : 2

Participant Entry 1
RMID  . . . . . : 1                   State . . . . . : 2

Participant Entry 2
RMID  . . . . . : 2                   State . . . . . : 2
```

Figure 35. Example dmpmqlog output (Part 9 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:57206>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:56282>

Version . . . . : 1


LOG RECORD - LSN <0:0:0:57440>
**********

HLG Header: lrecsize 224, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Forget (1539)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 204                 LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLFG
Version . . . . : 1                   Flags . . . . . : 0


LOG RECORD - LSN <0:0:0:58120>
**********

HLG Header: lrecsize 212, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Start Checkpoint (1025)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 192                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

No data for Start Checkpoint Record
```

Figure 35. Example dmpmqlog output (Part 10 of 12)

```
LOG RECORD - LSN <0:0:0:58332>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Transaction Table (773)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
TranCount . . . : 0


LOG RECORD - LSN <0:0:0:58552>
**********

HLG Header: lrecsize 1836, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Participants (1537)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 1816                LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPH
Version . . . . : 1                   Flags . . . . . : 3
Count . . . . . : 2

Participant Entry 0
RMName  . . . . : DB2 MQBankDB
RMID  . . . . . : 1
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :

Participant Entry 1
RMName  . . . . : DB2 MQFeeDB
RMID  . . . . . : 2
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :
```

Figure 35. Example dmpmqlog output (Part 11 of 12)

Example log file

```
LOG RECORD - LSN <0:0:0:60388>
**********

HLG Header: lrecsize 236, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM End Checkpoint (1026)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 216                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

ChkPtLSN  . . . : <0:0:0:58120>
OldestLSN . . . : <0:0:0:0>
MediaLSN  . . . : <0:0:0:0>


LOG RECORD - LSN <0:0:0:60624>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Stop Queue Manager (1028)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 220                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
StopDate  . . . : 19970519            StopTime  . . . : 10490868
SessionNumber . : 0                   ForceFlag . . . : Quiesce

AMQ7702: DMPMQLOG command has finished successfully.
```

Figure 35. Example dmpmqlog output (Part 12 of 12)

Notes for Figure 35:

1. The *headlsn* in the *Log File Header* has a value of <0:0:0:58120>. This is where the dump would start if we requested a different starting LSN.
2. The *nextlsn* is <0:0:0:60864>, the LSN of the first log record that the queue manager will write when it is next restarted.
3. The *HeadExtentID* is 1, indicating that the head of the log currently resides in log file S0000001.LOG.
4. The first log record formatted is a *Start Checkpoint* log record. The checkpoint spans a number of log records until the *End CheckPoint* record at <0:0:0:46448>.
5. One of the records logged during checkpoint is the *Transaction Participants* log record at <0:0:0:44594>. This details the resource managers that participate in global transactions coordinated by the queue manager.
6. The *Start Transaction* log record at <0:0:0:52262> denotes the start of a transaction. The *XTranid* shows a *TranType* of MQI, which indicates that it is a local transaction including WebSphere MQ updates only.
7. The next log record is a *Put Message* log record that records the persistent **MQPUT** under the syncpoint that started the transaction. The **MQPUT** was

made to the queue *Queue1* and the message data is logged as `Persistent message put under syncpoint`. This message has been allocated a *SpcIndex* of 1, which is matched to the later **MQGET** of this message.

8. The next log record at LSN <0:0:0:53458> is also a *Put Message* record. This persistent message was put to a different queue, *Queue2*, but was not made under syncpoint since the *XTranid* is *NULL*. It too has a *SpcIndex* of 1, which is a unique identifier for this particular queue.
9. The next log record at LSN <0:0:0:54192> commits the message that was put under syncpoint.
10. In log records <0:0:0:54408> and <0:0:0:54628> a new transaction is started by an **MQGET** under syncpoint for queue *Queue1*. The *SpcIndex* in the *Get Message* log record is 1 indicating that this was the same message that was put to *Queue1* in <0:0:0:52262>.
11. The next log record gets the message that was put to *Queue2* by the other *Put Message* log record.
12. The **MQGET** under syncpoint has been committed as indicated by the *Commit Transaction* log record at <0:0:0:55108>.
13. Finally an **MQBEGIN** is used to start a global transaction in the *Start Transaction* log record at <0:0:0:55324>. The *XTranid* in this log record has a *TranType* of XA.
14. The following *Put Message* records a persistent message put to *Queue2*. This shares the same *XTranid* as the previous log record.
15. If a *Transaction Prepared* log record is written for this *Xtranid* the transaction as a whole must be committed. The absence of such a log record can be taken as an indication that the transaction was rolled back. In this case a *Transaction Prepared* log record is found at <0:0:0:56498>. This records the queue manager itself as a participant with an *RMID* of zero. There are two further participants, their *RMIDs* of 1 and 2 can be matched with the previous *Transaction Participants* log record.
16. During the commit phase the XA Transaction Manager component of the queue manager does not log individual responses from the participants. The log indicates only whether the queue manager updates were committed or not. The *Commit Transaction* log record at <0:0:0:57206> indicates that the message was indeed committed to *Queue2*.
17. The *Transaction Forget* log record at <0:0:0:57440> indicates that the commit decision was also delivered to the other two resource managers. Any failure of these resource managers to commit their updates is diagnosed in the queue manager's error logs.

Example log file

Appendix I. 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	CICS	DB2
Encina	FFST	First Failure Support Technology
IBM	IBMLink	iSeries
MQSeries	OS/400	SupportPac
TXSeries	z/OS	zSeries

Lotus and Notes are trademarks of Lotus Development Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names, may be the trademarks or service marks of others.

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in U.S.A.

SC34-6068-01