WebSphere MQ for iSeries

System Administration Guide

Version 5 Release 3

SC34-6070-00

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix D, "Notices" on page 177.

First edition (October 2002)

This edition applies to WebSphere MQ for iSeries Version 5 Release 3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this book

This book applies to WebSphere® MQ for iSeries™ V5.3, the member of the WebSphere MQ family for the OS/400® operating system.

This product provides 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. The product uses 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 WebSphere MQ for iSeries, V5.3, and the services provided to support commercial messaging. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Installation of WebSphere MQ is described in *WebSphere MQ for iSeries 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 need to understand WebSphere MQ administration tasks.

What you need to know to understand this book

To use this book, you need a good understanding of the IBM® operating system for iSeries, and of the utilities associated with it. You do not need to have worked with message queuing products before, but you need to understand the basic concepts of message queuing.

For a summary of the new function introduced in WebSphere MQ for iSeries, V5.3. see “What’s new for this release” on page xiii.

For a list of the terms used in this book see the *WebSphere MQ Bibliography and Glossary*.

How to use this book

This book is divided into the following sections:

- The use of WebSphere MQ for iSeries using CL commands. This is the preferred method of operation.
- An overview of other methods of administering WebSphere MQ for iSeries, V5.3.
- The various features of the product.

About this book

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 for iSeries.

A change of name

With the release of Version 5 Release 3, we have rebranded MQSeries® to show its close relationship with IBM's 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.

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.

To help you to use *certificate revocation lists* (CRLs), WebSphere MQ uses a new object, the *authentication information object*. This book introduces the **WRKMQMAUTI**, **CHGMQMAUTI**, **CRTMQMAUTI**, **CPYMQMAUTI**, **DLTMQMAUTI**, and **DSPMQMAUTI** CL commands to support this object.

For a overview of SSL and the use of authentication information objects, see *WebSphere MQ Security*.

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 **GRTMQMAUT** commands against each individual object when it is created. "Generic OAM profiles" on page 59 provides detailed information on using generic profiles.

Modifications to existing commands

GRTMQMAUT and RVKMQMAUT have an additional parameter, `Service Component name`, that allows you to specify the name of the installed authorization service component. For more information, see "Specifying the installed authorization service" on page 60.

New commands for authority profiling

There are two new commands associated with authority profiling:

- WRKMQMAUT
- WRKMQMAUTD

For a full description of these, see "Working with authority profiles" on page 60

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 (**CHGMQMCAP**) and display (**DSPMQMCAP**) your license units.

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 for WebSphere MQ for iSeries was in *MQSeries Programmable System Management* We have now moved them into this book, in Chapter 10, "Installable services and components" on page 123.

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 "API exits" on page 116, 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*.

Quiescing a WebSphere MQ or MQSeries system

A new appendix, Appendix C, "Quiescing WebSphere MQ and MQSeries systems" on page 173, explains how to end gracefully (quiesce) a WebSphere MQ for iSeries or MQSeries for AS/400 system.

Chapter 1. Introduction to WebSphere MQ

This chapter introduces the WebSphere MQ for iSeries Version 5.3 product 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, OS/400 and z/OS™ applications can communicate through WebSphere MQ for iSeries 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) whatever platform the applications are run on. This makes it easier for you to port application programs from one platform to another.

The MQI is 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. In fact, the target application does not even have to be running when the message is sent. It can retrieve the message after it is has been started.

Message-driven processing

Upon arrival on a queue, messages can automatically start an application using a mechanism known as **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 for transferring information from one application program to

another (or to different parts of the same application). The applications can be running on the same platform, or on different platforms.

WebSphere MQ messages have two parts:

- **The application data**

  The content and structure of the application data is defined by the application programs that use them.
- **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*.

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 is 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 can 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. The messages are put on the queue by application programs, or by a **queue manager** as part of its normal operation.

Each queue is owned by a queue manager. The queue manager maintains the queues it owns and stores all the messages it receives onto the appropriate queues.

WebSphere MQ Version 5.3 supports queues over 2 GB in size; "Enabling large queues" on page 23 discusses this in more detail. For information about planning the amount of storage you need for queues, 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 that they are created:

- **Predefined queues** are created by an administrator using the appropriate WebSphere MQ script (MQSC) 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 OPEN request specifying the name of a **model queue**. The queue created is based on a *template queue definition*, which is the model queue. You can create a model queue using the MQSC DEFINE QMODEL command. 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**.

In WebSphere MQ, the object types include 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, or to achieve workload balancing. This is described in detail in *WebSphere MQ Queue Manager Clusters*.

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

- Chapter 2, “Managing WebSphere MQ for iSeries using CL commands” on page 13
- Chapter 3, “Alternative ways of administering WebSphere MQ” on page 31

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 referenced 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 “WebSphere MQ object names” on page 165.

Managing objects

You can manage objects using the native OS/400 menus.

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

- WebSphere MQ for iSeries CL commands
- WebSphere MQ script (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 more information about these methods, see Chapter 3, “Alternative ways of administering WebSphere MQ” on page 31.

You can also administer WebSphere MQ for iSeries from a Windows® machine using the WebSphere MQ Explorer (see “Using the WebSphere MQ Explorer” on page 35).

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 three ways of referring to an attribute:

- Using its CL parameter name, for example, MAXMSGLEN
- Using its PCF name, for example, *MaxMsgLength*.
- Using its MQSC name, for example, MAXMSGL.

The formal name of an attribute is its PCF name. Because using the CL interface is an important part of this book, you are more likely to see the CL name in examples than the PCF name of a given attribute.

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 exists on a remote machine across the network, or on the same machine as the local queue manager.

WebSphere MQ for iSeries 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.

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

WebSphere MQ queues

Queues are defined to WebSphere MQ using:

- The native OS/400 CRTMQMQ CL command
- The appropriate MQSC DEFINE command
- The PCF Create Queue command

Note: The WebSphere MQ process, channel, and namelist objects can be defined in a similar manner.

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*.

Using queue objects

There are four types of queue object available in WebSphere MQ. Each type of object can be manipulated by the product commands and is associated with real queues in different ways.

1. **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.
2. **A 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 *WebSphere MQ Queue Manager Clusters*.
3. **An 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 just change the alias queue definition to reflect the name of the new queue to which the alias resolves.

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

   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 are either temporary queues, which do not survive product restarts, or permanent queues, which do.

Specific local queue types and their uses

WebSphere MQ uses some local queues for specific purposes related to its operation. These are:

- **Application queues**

This is a queue that is used by an application through the MQI. It can be a local queue on the queue manager to which an application is linked, or it can be a remote queue that is owned by another queue manager.

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

- **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 up 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 26 For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

- **Transmission queues**

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

- **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 on every queue manager on WebSphere MQ for AIX®, iSeries, HP-UX, Solaris, and Windows.

  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.

  Cluster queue managers can communicate with queue managers that are not part of the cluster. To do this, the queue manager must define channels and a transmission queue to the other queue manager in the same way as in a traditional distributed-queuing environment.

  For more information on using clusters, see *WebSphere MQ Queue Manager Clusters*.

- **Dead-letter queues**

  A dead-letter 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. These queues are sometimes referred to as undelivered-message queues.

  A dead-letter queue is defined by default when each queue manager is created. However, you **must** ensure that the queue manager on which this queue resides points to the dead-letter queue that it is going to use.

  The following command creates an undelivered-message queue on queue manager neptune.queue.manager:

  ```
  CRTMQM MQMNAME(neptune.queue.manager) UDLMSGQ(ANOTHERDLQ)
  ```

- **Command queues**

The command queue, named SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send WebSphere MQ 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**

  WebSphere MQ for iSeries supports instrumentation events, which can be used to monitor queue managers independently of MQI applications. Instrumentation events can be generated in several ways, for example:

  – An application attempting to put a message on a queue that is not available or does not exist.
  – A queue becoming full.
  – A channel being started.

  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 informs 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*.

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on a WebSphere MQ queue manager. See the "Initiation queues" entry under "Specific local queue types and their uses" on page 6 for more information.

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

Use the WebSphere MQ for iSeries CRTMQMPRC CL command, the MQSC command DEFINE PROCESS, or the PCF command Create Process to create a process definition.

Channels

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

Use the WebSphere MQ for iSeries CRTMQMCHL CL command, the MQSC command DEFINE CHANNEL, or the PCF command Create Channel to create a channel definition.

Note: Clustering automates some of these tasks for you.

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 have defined 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 complex transmission queue, channel, and queue definitions.

For information about clusters, see *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; that is, 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 so that you can maintain a list of clusters referenced by more than one WebSphere MQ object.

Use the WebSphere MQ for iSeries CRTMQMNL CL command, the MQSC command DEFINE NAMELIST, or the PCF command Create Namelist to create a namelist definition.

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 introduces the **WRKMQMAUTI**, **DSPMQMAUTI**, **CRTMQMAUTI**, **CPYMQMAUTI**, **CHGMQMAUTI**, and **DLTMQMAUTI** commands for use with the authentication information object. For an overview of SSL and the use of 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.DEF; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE, and the default receiver channel is SYSTEM.DEF.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.

Clients and servers

WebSphere MQ supports client-server configurations for WebSphere MQ applications.

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

Note: WebSphere MQ for iSeries acts as a Java™ client only.

An *MQI 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, that is, on the MQI server machine. A server can support normal local WebSphere MQ applications as well.

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.

The advantages of a client are that:
- It is simple to set up
- It is simple to manage
- It has a low resource footprint

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 defining user exits.

User exits

User exits provide a way 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*. The exit is supported in the following environments: AIX, iSeries, HP-UX, Solaris, Windows, and z/OS

Security

In WebSphere MQ for iSeries security is provided by the Object Authority Manager (OAM) component. See Chapter 5, “Protecting WebSphere MQ objects” on page 47 for details of this component.

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.

A unit of work **commits** when it completes successfully. At this point all updates made within that unit of work are made permanent or irreversible. If the unit of work fails, all updates are instead *backed out*. *Syncpoint coordination* is the process by which units of work are either committed or backed out with integrity.

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 dual-phase commit process and the MQI calls MQBACK and MQCMIT.

WebSphere MQ for iSeries can support and participate in global units of work as a resource manager, coordinated by the OS/400 COMMIT and ROLLBACK commands.

Chapter 2. Managing WebSphere MQ for iSeries using CL commands

WebSphere MQ for iSeries provides three interfaces for administration:

- A set of fully featured OS/400 commands
- Provision for processing the WebSphere MQ script (MQSC) commands. This is a common interface used by all WebSphere MQ implementations on other platforms
- WebSphere MQ programmable command formats (PCF). This is a lower-level command interface, which can be used by applications to administer the product.

This chapter gives an overview of working with WebSphere MQ for iSeries using the WebSphere MQ OS/400 commands, together with some suggested operations, as these commands are specific to iSeries.

WebSphere MQ applications

When you create or customize WebSphere MQ applications, it is useful to keep a record of all WebSphere MQ definitions created. This record can be used for:

- Recovery purposes
- Maintenance
- Rolling out WebSphere MQ applications

You can do this by either:

- Creating CL programs to generate your WebSphere MQ definitions for the server, or
- Creating MQSC text files as SRC members to generate your WebSphere MQ definitions using the cross-platformWebSphere MQ command language.

WebSphere MQ for iSeries CL commands

The commands can be grouped as follows:

- Authentication Information Commands
  - CHGMQMAUTI, Change MQM Authentication Information
  - CPYMQMAUTI, Copy MQM Authentication Information
  - DLTMQMAUTI, Delete MQM Authentication Information
  - CRTMQMAUTI, Create MQM Authentication Information
  - DSPMQMAUTI, Display MQM Authentication Information
  - WRKMQMAUTI, Work with MQM Authentication Information
- Channel Commands
  - CHGMQMCHL, Change MQM Channel
  - CPYMQMCHL, Copy MQM Channel
  - CRTMQMCHL, Create MQM Channel
  - DLTMQMCHL, Delete MQM Channel
  - DSPMQMCHL, Display MQM Channel
  - ENDMQMCHL, End MQM Channel
  - ENDMQMLSR, End MQM Listener
  - PNGMQMCHL, Ping MQM Channel
  - RSTMQMCHL, Reset MQM Channel
  - RSVMQMCHL, Resolve MQM Channel

CL commands

STRMQMCHL, Start MQM Channel
STRMQMCHLI, Start MQM Channel Initiator
STRMQMLSR, Start MQM Listener
WRKMQMCHL, Work with MQM Channel
WRKMQMCHST, Work with MQM Channel Status
WRKMQMLSR, Work with MQM Listener

- Cluster Commands
  RFRMQMCL, Refresh Cluster
  RSMMQMCLQM, Resume Cluster Queue Manager
  RSTMQMCL, Reset Cluster
  SPDMQMCLQM, Suspend Cluster Queue Manager
  WRKMQMCL, Work with Clusters
  WRKMQMCLQM, Work with Cluster Queue Manager
- Command Server Commands
  DSPMQMCSVR, Display MQM Command Server
  ENDMQMCSVR, End MQM Command Server
  STRMQMCSVR, Start MQM Command Server
- Data Type Conversion Command
  CVTMQMDTA, Convert MQM Data Type Command
- Dead-Letter Queue Handler Command
  STRMQMDLQ, Start WebSphere MQ Dead-Letter Queue Handler
- License Unit Commands
  DSPMQMCAP, Display License Units
  CHGMQMCAP, Change License Units
- Media Recovery Commands
  RCDMQMIMG, Record MQM Object Image
  RCRMQMOBJ, Recreate MQM Object
- Name Command
  DSPMQMOBJN, Display MQM Object Names
- Namelist Commands
  CHGMQMNL, Change MQM Namelist
  CPYMQMNL, Copy MQM Namelist
  CRTMQMNL, Create MQM Namelist
  DLTMQMNL, Delete MQM Namelist
  DSPMQMNL, Display MQM Namelist
  WRKMQMNL, Work with MQM Namelists
- Process Commands
  CHGMQMPRC, Change MQM Process
  CPYMQMPRC, Copy MQM Process
  CRTMQMPRC, Create MQM Process
  DLTMQMPRC, Delete MQM Process
  DSPMQMPRC, Display MQM Process
  WRKMQMPRC, Work with MQM Processes
- Queue Commands
  CHGMQMQ, Change MQM Queue
  CLRMQMQ, Clear MQM Queue
  CPYMQMQ, Copy MQM Queue
  CRTMQMQ, Create MQM Queue
  DLTMQMQ, Delete MQM Queue
  DSPMQMQ, Display MQM Queue
  WRKMQMMSG, Work with MQM Messages
  WRKMQMQ, Work with MQM Queues
  WRKMQMQSTS, Work with MQM Queue Status

- Queue Manager Commands
  - CCTMQM, Connect to Message Queue Manager
  - CHGMQM, Change Message Queue Manager
  - CRTMQM, Create Message Queue Manager
  - DLTMQM, Delete Message Queue Manager
  - DSCMQM, Disconnect from Message Queue Manager
  - DSPMQM, Display Message Queue Manager
  - ENDMQM, End Message Queue Manager
  - STRMQM, Start Message Queue Manager
  - WRKMQM, Work with Message Queue Manager
- Security Commands
  - DSPMQMAUT, Display MQM Object Authority
  - GRTMQMAUT, Grant MQM Object Authority
  - RFRMQMAUT, Refresh Security
  - RVKMQMAUT, Revoke MQM Object Authority
  - WRKMQMAUT, Work with MQM Authority
  - WRKMQMAUTD, Work with MQM Authority Data
- Trace Commands
  - TRCMQM, Trace MQM Job
- Transaction Commands
  - STRMQMTRN, Start WebSphere MQ Transaction
  - WRKMQMTRN, Work with WebSphere MQ Transactions
  - RSVMQMTRN, Resolve WebSphere MQ Transaction
- Trigger Monitor Command
  - STRMQMTRM, Start Trigger Monitor
- WebSphere MQ Commands
  - RUNMQSC, Run MQSC Commands
  - STRMQMMQSC, Start MQSC Commands

General usage tips

Most groups of WebSphere MQ commands, including those associated with queue managers, queues, channels, namelists, process definitions, and authentication information objects can be accessed using the relevant WRK* command.

The principal command in the set is WRKMQM. This command allows you, for example, to display a list of all the queue managers on the system, together with status information. Alternatively, you can process all queue-manager specific commands using various options against each entry.

From the WRKMQM command you can select specific areas of each queue manager, for example, working with channels or queues, and from there select individual objects.

Before you start

Ensure that the WebSphere MQ subsystem is running (using the command `STRSBS QMQM/QMQM`), and that the job queue associated with that subsystem is not held. By default, the WebSphere MQ subsystem and job queue are both named QMQM in library QMQM.

Starting a local queue manager

You must:

Starting local queue manager

1. Create a local queue manager by issuing the **CRTMQM** command from an OS/400 command line.

   When you create a queue manager, you have the option of making that queue manager the default queue manager.

   The default queue manager (of which there can only be one) is the queue manager to which a CL command applies, if the queue manager name (MQMNAME) parameter is omitted.

   Note: One queue manager **must** be selected as the default queue manager.
2. Start a local queue manager by issuing the **STRMQM** command from an OS/400 command line.

You can stop a queue manager by issuing the **ENDMQM** command from the OS/400 command line, and control a queue manager by issuing other WebSphere MQ commands from an OS/400 command line.

The principal commands are described later in this chapter.

Remote queue managers cannot be started remotely but must be created and started in their systems by local operators. An exception to this is where remote operating facilities (outside WebSphere MQ for iSeries) exist to enable such operations.

The local queue administrator cannot stop a remote queue manager.

Note: As part of quiescing a WebSphere MQ (or MQSeries) system, you have to quiesce the active queue managers. This is described in Appendix C, "Quiescing WebSphere MQ and MQSeries systems" on page 173.

Creating WebSphere MQ objects

The following tasks suggest various ways in which you can use WebSphere MQ for iSeries from the command line.

There are two online methods to create WebSphere MQ objects, which are:

1. Using a Create command
   CRTMQMAUTI
   Create MQM Authentication Information Object
   CRTMQMCHL
   Create MQM Channel
   CRTMQMNL
   Create MQM Namelist
   CRTMQMPRC
   Create MQM Process
   CRTMQMQ
   Create MQM Queue
2. Using the appropriate Work with MQM object command, followed by F6:
   WRKMQMAUTI
   Work with MQM Authentication Information Object
   WRKMQMCHL
   Work with MQM Channels
   WRKMQMNL
   Work with MQM Namelists
   WRKMQMPRC
   Work with MQM Processes

WRKMQMQ
Work with MQM Queues

Note: All MQM commands can be submitted from the Message Queue Manager Commands menu. To display this menu, type `GO CMDMQM` on the command line and press the `Enter` key.

The system displays the prompt panel automatically when you select a command from this menu. To display the prompt panel for a command that you have typed directly on the command line, press `F4` before pressing the `Enter` key.

Examples of creating a local queue

To create a local queue from the command line, use:
1. The Create MQM Queue (**CRTMQMQ**) command
2. The Work with MQM Queues (**WRKMQMQ**) command, followed by F6

Creating a local queue using the CRTMQMQ command

1. Type **CRTMQMQ** on the command line and press the `F4` key.
2. On the Create MQM Queue panel, type the name of the queue that you want to create in the `Queue name` field.

   To specify a mixed case name, you enclose the name in apostrophes.
3. Type `*LCL` in the `Queue type` field.
4. Specify a queue manager name, unless you are using the default queue manager, and press the `Enter` key. Further settings for a local queue are displayed (see Figure 1) with the fields containing the default values. You can overtype any of these values with a new value.

   Scroll forward to see further fields. The options used for clusters are at the end of the list of options.
5. When you have changed any values, press the `Enter` key to create the queue.

```
                        Create MQM Queue (CRTMQMQ)

Type choices, press Enter.

Queue name . . . . . . . . . . . > TEST.QUEUE.LCL

Queue type . . . . . . . . . . . > *LCL          *ALS, *LCL, *MDL, *RMT
Message Queue Manager Name . . .   MY.QUEUE.MANAGER_____________________________
_____________________________
Replace  . . . . . . . . . . . .   *NO_          *NO, *YES
Text 'description' . . . . . . .   '_____________________________________________
_____________________________      '
Put enabled  . . . . . . . . . .   *YES____      *SYSDFTQ, *NO, *YES
Default message priority . . . .   5_________    0-9, *SYSDFTQ
Default message persistence  . .   *NO_____      *SYSDFTQ, *NO, *YES
Process name . . . . . . . . . .   '_____________________________________________
_____
Triggering enabled . . . . . . .   *NO_____      *SYSDFTQ, *NO, *YES
Get enabled  . . . . . . . . . .   *YES____      *SYSDFTQ, *NO, *YES
Sharing enabled  . . . . . . . .   *YES____      *SYSDFTQ, *NO, *YES
                                                                      More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 1. Create MQM Queue initial panel

Creating a local queue using the WRKMQMQ command

1. Type **WRKMQMQ** on the command line.
2. Enter the name of a queue manager
3. If you want to display the prompt panel, press F4.

   The prompt panel is useful to reduce the number of queues displayed, by specifying a generic queue name or queue type.
4. Press the Enter key and Figure 2 is displayed.

```
                            Work with MQ Queues

Queue Manager Name . . :   mick

Type options, press Enter.
  2=Change   3=Copy   4=Delete   5=Display   12=Work with messages ...


Opt  Name                                              Type  Depth   Jobs
     SYSTEM.ADMIN.CHANNEL.EVENT                        *LCL      0      0
     SYSTEM.ADMIN.COMMAND.QUEUE                        *LCL      0      0
     SYSTEM.ADMIN.PERFM.EVENT                          *LCL      0      0
     SYSTEM.ADMIN.QMGR.EVENT                           *LCL      1      0
     SYSTEM.AUTH.DATA.QUEUE                            *LCL     29      1
     SYSTEM.CHANNEL.INITQ                              *LCL      0      1
     SYSTEM.CHANNEL.SYNCQ                              *LCL      1      0
     SYSTEM.CICS.INITIATION.QUEUE                      *LCL      0      0
     SYSTEM.CLUSTER.COMMAND.QUEUE                      *LCL      0      1
                                                                    More...
Parameters for options 2, 3, 4, 5, 12, 13, 14, 15, 16 or command
===>
F3=Exit   F4=Prompt   F5=Refresh   F6=Create   F9=Retrieve   F11=Change View
F12=Cancel   F16=Repeat position to   F23=More options       F24=More keys
```

Figure 2. Work with MQM Queues panel

5. Press F6 to create a new queue; this takes you to the **CRTMQMQ** panel. See "Creating a local queue using the CRTMQMQ command" on page 17 for instructions on how to create the queue.

   When you have created the queue, the Work with MQM Queues panel is displayed again. The new queue is added to the list when you press F5=Refresh.

Examples of creating a remote queue

Use the CRTMQMQ panel to define the queue with queue type *RMT, using one of the following online methods:

1. The **CRTMQMQ** command.
2. F6=Create on the **WRKMQMQ** panel.

The use of remote queues is described in detail in *WebSphere MQ Intercommunication*.

This section describes how to define a remote queue for each of the three uses. We use the CRTMQMQ command in the examples; you can, of course, do the same thing from the WRKMQMQ panel.

Creating a remote queue as a remote queue definition

This is the most straightforward use of remote queues. It is used to direct messages to a local queue on a remote queue manager, through a transmission queue.

To create a remote queue for this use:

1. Type **CRTMQMQ** on the command line and press the F4 key.
2. Type the queue name in the Queue name field.
3. Type *RMT in the Queue type field.
4. Type the name of the local queue manager in the Queue Manager Name field.
5. Type the name of the local queue at the remote location in the Remote queue field.
6. Type the name of the queue manager at the remote location in the Remote Message Queue Manager field.
7. Optionally, type the name of the transmission queue to the remote location in the Transmission queue field.

   If you do not specify a transmission queue name, the transmission queue with the same name as the remote queue manager is used.

Creating a remote queue as a queue manager alias

Queue manager alias definitions can be used to remap the queue manager name specified in the MQOPEN call. This enables you to alter the target queue manager without changing your applications.

See *WebSphere MQ Intercommunication* for further information.

To define a remote queue as a queue manager object:

1. Type **CRTMQMQ** on the command line and press the F4 key.
2. Type the queue name in the Queue name field.
3. Type *RMT in the Queue type field.
4. Type the name of the local queue manager in the Queue Manager Name field.
5. Type the name of the queue manager at the remote location in the Remote Message Queue Manager field.
6. Optionally, type the name of the transmission queue to the remote location in the Transmission queue field.

   If you do not specify a transmission queue name, the transmission queue with the same name as the remote queue manager is used.

Creating a remote queue as an alias to a reply-to queue

An application can name a reply-to queue when it puts a message on a queue. The reply-to queue name is used by the application that gets the message from the queue to send reply messages. To define an alias to a reply-to queue, define a remote queue with the same name as the reply-to queue.

See *WebSphere MQ Intercommunication* for further information.

To create a remote queue as an alias to a reply-to queue:

1. Type **CRTMQMQ** on the command line and press the F4 key.
2. Type the queue name in the Queue name field.

   This must be the same as the reply-to queue named by the putting application.
3. Type *RMT in the Queue type field.
4. Type the name of the local queue manager in the Queue Manager Name field, unless you are using the default queue manager.
5. Type the queue name in the Queue name field.

   This is the name of the queue to which you want the reply-to messages sent.
6. Type the name of the queue manager at the remote location in the Remote Message Queue Manager field.

This is the name of the queue manager to which you want the reply-to messages sent.

7. Optionally, type the name of the transmission queue to the remote location in the `Transmission queue` field.

   If you do not specify a transmission queue name, the transmission queue with the same name as the remote queue manager is used.

Creating a transmission queue

A transmission queue is a local queue that is used to send messages to a remote queue manager, through a message channel, which provides a one-way link to the remote queue manager.

Each message channel has a transmission queue name specified at the sending end of the message channel.

Note: If you use clusters, you do not have to create a transmission queue.

To create a transmission queue:

1. Type **CRTMQMQ** on the command line and press the `F4` key.
2. Type the queue name in the `Queue name` field.

   If you want to define a default transmission queue for all messages destined to a remote queue manager, the transmission queue name must be the same as the remote queue manager name.
3. Type `*LCL` in the `Queue type` field.
4. Type `*TMQ` in the `Usage` field.

Creating an initiation queue

An initiation queue is a local queue on which the queue manager puts trigger messages in response to a trigger event, for example, a message arriving on a local queue. An initiation queue is a local queue and has no special settings that define it as an initiation queue.

For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

Creating an alias queue

Use an alias queue object to access another queue on the local queue manager. Any messages put on the alias queue are redirected to the queue named in the alias queue definition.

Note: An alias queue cannot hold messages itself.

To create an alias queue:

1. Type **CRTMQMQ** on the command line and press the `F4` key.
2. Type the queue name in the `Queue name` field.
3. Type `*ALS` in the `Queue type` field.
4. Type the name of the local queue manager in the `Queue Manager Name` field.
5. Type the name of the local queue that you want the queue name to resolve to in the `Target queue` field.

Creating a model queue

Define a model queue with a set of attributes in the same way that you define a local queue. Type `*MDL` in the `Queue type` field.

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.)

Altering queue manager attributes

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

```
CHGMQM MQMNAME('jupiter.queue.manager') UDLMSGQ(ANOTHERDLQ) INHEVT(*YES)
```

This command changes the dead-letter queue used, and enables inhibit events.

Working with local queues

This section contains examples of some of the commands that you can use to manage local, model, and alias queues. All the commands shown are also available using options from the WRKMQMQ command panel.

Defining a local queue

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

Use the command CRTMQMQ QTYPE *LCL to create a definition of a local queue and also to create the data structure that is called a queue. You can also modify the queue characteristics from those of the default local queue.

In this example, the queue we define, ORANGE.LOCAL.QUEUE, is specified to have these characteristics:

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

The following command does this on the default queue manager:

```
CRTMQMQ QNAME('orange.local.queue') QTYPE(*LCL)
       TEXT('Queue for messages from other systems')
       PUTENBL(*NO)
       GETENBL(*YES)
       TRGENBL(*NO)
       MSGDLYSEQ(*FIFO)
       MAXDEPTH(1000)
       MAXMSGLEN(2000)
       USAGE(*NORMAL)
```

Notes:

1. USAGE *NORMAL indicates that this queue is not a transmission queue.

2. If you already have a local queue on the same queue manager with the name orange.local.queue, this command fails. Use the REPLACE *YES attribute, if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 23.

Defining a dead-letter queue

Each queue manager must have a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You must explicitly tell the queue manager about the dead-letter queue. You can do this by specifying a dead-letter queue on the **CRTMQM** command, or you can use the **CHGMQM** command to specify one later. You must also define the dead-letter queue before it can be used.

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is supplied with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required. There is no need to rename it, although you can if you like.

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

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

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 6, "The WebSphere MQ dead-letter queue handler" on page 69.

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:

```
DSPMQMQ MQMNAME(MYQUEUEMANAGER) QNAME(SYSTEM.DEFAULT.LOCAL.QUEUE)
```

Copying a local queue definition

You can copy a queue definition using the **CPYMQMQ** command. For example:

```
CPYMQMQ MQMNAME(MYQUEUEMANAGER) FROMQ('orange.local.queue') TOQ('magenta.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.

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

```
CPYMQMQ MQMNAME(MYQUEUEMANAGER) FROMQ('orange.local.queue') TOQ('third.queue')
  MAXMSGLEN(1024)
```

This command copies the attributes of the queue orange.local.queue to the queue third.queue, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Note: When you use the **CPYMQMQ** command, you copy the queue attributes only, not the messages on the queue.

Changing local queue attributes

You can change queue attributes in two ways, using either the **CHGMQMQ** command or the **CPYMQMQ** command with the REPLACE *YES attribute. In "Defining a local queue" on page 21, we defined the queue orange.local.queue. Suppose, for example, you wanted to increase the maximum message length on this queue to 10 000 bytes.

- Using the **CHGMQMQ** command:

```
CHGMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('orange.local.queue') MAXMSGLEN(10000)
```

  This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.

- Using the **CRTMQMQ** command with the REPLACE *YES option, for example:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('orange.local.queue') QTYPE(*LCL) MAXMSGLEN(10000)
  REPLACE(*YES)
```

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

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

Clearing a local queue

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

```
CLRMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('magenta.queue')
```

You cannot clear a queue if:

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

Deleting a local queue

Use the command DLTMQMQ to delete a local queue. A queue cannot be deleted if it has uncommitted messages on it.

Enabling large queues

WebSphere MQ Version 5.3 supports queues larger than 2 GB. See your operating system documentation for information on how to enable OS/400 to support large files.

Some utilities might not be able to cope with files greater than 2 GB. Before enabling large file support, check your operating system documentation for information on restrictions on such support.

Working with alias queues

An alias queue (sometimes known as a queue alias) provides a method of redirecting MQI calls. An alias queue is not a real queue but a definition that resolves to a real queue. The alias queue definition contains a target queue name, which is specified by the TGTQNAME attribute.

When an application specifies an alias queue in an MQI call, the queue manager resolves the real queue name at run time.

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

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

Defining an alias queue

The following command creates an alias queue:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('my.alias.queue') QTYPE(*ALS) TGTQNAME('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:

```
CHGMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('my.alias.queue') TGTQNAME('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.

You can do this using the following commands:

```
/* This alias is put enabled and get disabled for application ALPHA */

CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('alphas.alias.queue') QTYPE(*ALS)
  TGTQNAME('yellow.queue') PUTENBL(*YES) GETENBL(*NO)

/* This alias is put disabled and get enabled for application BETA */

CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('betas.alias.queue') QTYPE(*ALS)
  TGTQNAME('yellow.queue') PUTENBL(*NO) GETENBL(*YES)
```

ALPHA uses the queue name alphas.alias.queue in its MQI calls; BETA uses the queue name betas.alias.queue. They both access the same queue, but in different ways.

You can use the REPLACE *YES attribute when you define alias queues, in the same way that you use these attributes with local queues.

Using other commands with alias queues

You can use the appropriate commands to display or change alias queue attributes. For example:

```
 * Display the alias queue's attributes */

DSPMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('alphas.alias.queue')

/* ALTER the base queue name, to which the alias resolves. */
/* FORCE = Force the change even if the queue is open. */

CHQMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('alphas.alias.queue')
      TGTQNAME('orange.local.queue') FORCE(*YES)
```

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 they are required.

Defining a model queue

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

```
CRTMQMQ QNAME('green.model.queue') QTYPE(*MDL) DFNTYPE(*PERMDYN)
```

This command creates a model queue definition. From the DFNTYPE attribute, the actual queues created from this template are permanent dynamic queues. The attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the REPLACE *YES attribute when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

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

```
/* Display the model queue's attributes */

DSPMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('green.model.queue')

/* ALTER the model queue to enable puts on any */
/* dynamic queue created from this model. */

CHGMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('blue.model.queue') PUTENBL(*YES)
```

Managing objects for triggering

WebSphere MQ provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *WebSphere MQ Application Programming Guide*.

This section describes 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 TRGENBL attribute.

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

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.insurance.queue') QTYPE(*LCL)
       PRCNAME('motor.insurance.quote.process') MAXMSGLEN(2000)
       DFTMSGPST(*YES) INITQNAME('motor.ins.init.queue')
       TRGENBL(*YES) TRGTYPE(*DEPTH) TRGDEPTH(100) TRGMSGPTY(5)
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
The name of the queue manager.

QNAME('motor.insurance.queue')
The name of the application queue being defined.

PRCNAME('motor.insurance.quote.process')
The name of the application to be started by a trigger monitor program.

MAXMSGLEN(2000)
The maximum length of messages on the queue.

DFTMSGPST(*YES)
Messages on this queue are persistent by default.

INITQNAME('motor.ins.init.queue')
The name of the initiation queue on which the queue manager is to put the trigger message.

TRGENBL(*YES)
The trigger attribute value.

TRGTYPE(*DEPTH)
A trigger event is generated when the number of messages of the required priority (TRGMSGPTY) reaches the number specified in TRGDEPTH.

TRGDEPTH(100)
The number of messages required to generate a trigger event.

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

Defining an initiation queue

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

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.ins.init.queue') QTYPE(*LCL)
       GETENBL(*YES) SHARE(*NO) TRGTYPE(*NONE)
       MAXMSGL(2000)
       MAXDEPTH(1000)
```

Creating a process definition

Use the CRTMQMPRC command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PRCDEFN attribute on the application queue motor.insurance.queue. The following command creates the required process, motor.insurance.quote.process, identified in this example:

```
CRTMQMPRC MQMNAME(MYQUEUEMANAGER) PRCNAME('motor.insurance.quote.process')
          TEXT('Insurance request message processing')
          APPTYPE(*OS400) APPID(MQTEST/TESTPROG)
          USRDATA('open, close, 235')
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
The name of the queue manager.

PRCNAME('motor.insurance.quote.process')
The name of the process definition.

TEXT('Insurance request message processing')
A description of the application program to which this definition relates. This text is displayed when you use the DSPMQMPRC command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPTYPE(*OS400)
The type of application to be started.

APPID(MQTEST/TESTPROG)
The name of the application executable file, specified as a fully qualified file name.

USRDATA('open, close, 235')
User-defined data, which can be used by the application.

Displaying your process definition

Use the **DSPMQMPRC** command to examine the results of your definition. For example:

```
MQMNAME(MYQUEUEMANAGER) DSPMQMPRC('motor.insurance.quote.process')
```

You can also use the **CHGMQMPRC** command to alter an existing process definition, and the **DLTMQMPRC** command to delete a process definition.

Communicating between two systems

The following example illustrates how to set up two WebSphere MQ for iSeries systems, using CL commands, so that they can communicate with one another.

Distributed queuing example

The systems are called SYSTEMA and SYSTEMB, and the communications protocol used is TCP/IP.

Carry out the following procedure:

1. Create a queue manager on SYSTEMA, calling it QMGRA1.

```
CRTMQM     MQMNAME(QMGRA1) TEXT('System A - Queue +
             Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
```

2. Start this queue manager.

```
STRMQM     MQMNAME(QMGRA1)
```

3. Define the WebSphere MQ objects on SYSTEMA that you need to send messages to a queue manager on SYSTEMB.

```
/* Transmission queue */
 CRTMQMQ    QNAME(XMITQ.TO.QMGRB1) QTYPE(*LCL) +
              MQMNAME(QMGRA1) TEXT('Transmission Queue +
              to QMGRB1') MAXDEPTH(5000) USAGE(*TMQ)

/* Remote queue that points to a queue called TARGETB               */
/* TARGETB belongs to queue manager QMGRB1 on SYSTEMB               */
 CRTMQMQ    QNAME(TARGETB.ON.QMGRB1) QTYPE(*RMT) +
              MQMNAME(QMGRA1) TEXT('Remote Q pointing +
              at Q TARGETB on QMGRB1 on Remote System +
              SYSTEMB') RMTQNAME(TARGETB) +
              RMTMQMNAME(QMGRB1) TMQNAME(XMITQ.TO.QMGRB1)

/* TCP/IP sender channel to send messages to the queue manager on SYSTEMB*/
 CRTMQMCHL  CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*SDR) +
              MQMNAME(QMGRA1) TRPTYPE(*TCP) +
              TEXT('Sender Channel From QMGRA1 on +
              SYSTEMA to QMGRB1 on SYSTEMB') +
              CONNAME(SYSTEMB) TMQNAME(XMITQ.TO.QMGRB1)
```

4. Create a queue manager on SYSTEMB, calling it QMGRB1.

```
CRTMQM     MQMNAME(QMGRB1) TEXT('System B  - Queue +
             Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
```

5. Start the queue manager on SYSTEMB.

```
STRMQM     MQMNAME(QMGRB1)
```

6. Define the WebSphere MQ objects that you need to receive messages from the queue manager on SYSTEMA.

```
/* Local queue to receive messages on */
 CRTMQMQ    QNAME(TARGETB) QTYPE(*LCL) MQMNAME(QMGRB1) +
              TEXT('Sample Local Queue for QMGRB1')

/* Receiver channel of the same name as the sender channel on SYSTEMA */
 CRTMQMCHL  CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*RCVR) +
              MQMNAME(QMGRB1) TRPTYPE(*TCP) +
              TEXT('Receiver Channel from QMGRA1 to +
              QMGRB1')
```

7. Finally, start a TCP/IP listener on SYSTEMB so that the channel can be started. This example uses the default port of 1414.

```
STRMQMLSR MQMNAME(QMGRB1)
```

You are now ready to send test messages between SYSTEMA and SYSTEMB. Using one of the supplied samples, put a series of messages to your remote queue on SYSTEMA.

Start the channel on SYSTEMA, either by using the command **STRMQMCHL**, or by using the command **WRKMQMCHL** and entering a start request (Option 14) against the sender channel.

The channel should go to RUNNING status and the messages are sent to queue TARGETB on SYSTEMB.

Check your messages by issuing the command:

```
WRKMQMMSG QNAME(TARGETB) MQMNAME(QMGRB1).
```

Distributed queuing example

Chapter 3. Alternative ways of administering WebSphere MQ

You normally use OS/400® CL commands to administer WebSphere MQ for iSeries. See Chapter 2, "Managing WebSphere MQ for iSeries using CL commands" on page 13 for an overview of these commands.

Using CL commands is the preferred method of administering the system. However, you can use various other methods. This chapter gives an overview of those methods and includes the following topics:

Local and remote administration

You administer WebSphere MQ objects locally or remotely.

Local administration means carrying out administration tasks on any queue managers that you have defined on your local system. In WebSphere MQ, you can consider this as local administration because no WebSphere MQ channels are involved, that is, the communication is managed by the operating system. 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.

WebSphere MQ supports administration from a single point through what is known as *remote administration*. Remote administration consists of sending programmable command format (PCF) control messages to the SYSTEM.ADMIN.COMMAND.QUEUE on the target queue manager.

There are a number of ways of generating PCF messages. These are:

1. Writing a program using PCF messages. See "Administration using PCF commands" on page 33.
2. Writing a program using the MQAI, which sends out PCF messages. See "Using the MQAI to simplify the use of PCFs" on page 34.
3. Using the WebSphere MQ Explorer, available with WebSphere MQ for Windows, which allows you to use a graphical user interface (GUI) and generates the correct PCF messages. See "Using the WebSphere MQ Explorer" on page 35.

For example, you can issue a remote command to change a queue definition on a remote queue manager.

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

Administration using MQSC commands

You use WebSphere MQ script (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 STRMQMMQSC WebSphere MQ CL command. This is a batch method only, taking its input from a SRC PHYSICAL file in the server library system. The default name for this source physical file is QMQSC.

WebSphere MQ for iSeries does not supply a source file called QMQSC. To process MQSC commands you need to create the QMQSC source file in a library of your choice, by issuing the following command:

```
CRTSRCPRF FILE(MYLIB/QMQSC) TEXT('WebSphere MQ - MQSC Source')
```

MQSC source is held in members within this source file. To work with the members enter the following command:

```
WRKMBRPDM MYLIB/QMQSC
```

You can now add new members and maintain existing ones

You can also enter MQSC commands interactively, by:

1. Typing in the queue manager name and pressing the `Enter` key to access the WRKMQM results panel
2. Selecting `F23=More options` on this panel
3. Selecting option 26 against an active queue manager on the panel shown in Figure 3

To end such an MQSC session, type **end**.

```
                          Work with queue managers

Type options, press Enter.
  21=Work with NameLists     22=Work with Jobs       23=Display logs
  24=Work with Authorities   25=Work with AuthInfo   26=MQSC ...

Opt  Name                                                  Status  Default
     mick                                                  ACTIVE     NO
     mike                                                  ACTIVE     NO

                                                                      Bottom

Parameters or command
===>
F3=Exit   F4=Prompt      F5=Refresh   F6=Create   F9=Retrieve   F12=Cancel
F16=Repeat position to   F17=Position to   F23=More options     F24=More keys
```

Figure 3. Work with queue managers results panel

MQSC command files

MQSC commands are written in human-readable form, that is, in EBCDIC text.

Figure 4 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes.

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

Figure 4. Extract from the MQSC command file, myprog.in

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

Object attributes specified in MQSC are shown in this book in uppercase (for example, RQMNAME), although they are not case sensitive.

Notes:

1. The format of an MQSC file does not depend on its location in the file system
2. MQSC attribute names are limited to eight characters.
3. MQSC commands are available on other platforms, including z/OS®.

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

Administration using 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 and process definitions, and change queue managers, from a program.

PCF commands cover the same range of functions provided by MQSC commands. However, unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of 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.

Briefly, these are some of the things the application programmer must specify 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*.

Attributes in MQSC and PCF commands

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

Object attributes in PCF commands, which are not limited to eight characters, 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

You can use the WebSphere MQ Administration Interface (MQAI) to obtain easier programming access to PCF messages.

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 and this avoids the problems associated with complex data structures.

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

On the other hand, programs written using the MQAI pass parameters into the appropriate data bag and only one statement is required 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.

Handle error conditions more easily
It is difficult to get return codes back from MQSC 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 then 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 the *WebSphere MQ Programmable Command Formats and Administration Interface*.

Using the WebSphere MQ Explorer

The WebSphere MQ Explorer is an application that runs under the Microsoft® Management Console (MMC) on Windows NT version 4.0, Windows 2000, and Windows XP. It provides a graphical user interface for controlling WebSphere MQ resources in a WebSphere MQ network.

The platforms and levels of WebSphere MQ that can be administered using the WebSphere MQ Explorer are described in "Prerequisite software" on page 36.

Using the online guidance, you can:

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

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

The configuration steps you must perform on remote WebSphere MQ queue managers to allow the WebSphere MQ Explorer to administer them are outlined in "Required definitions for administration" on page 36.

This section contains the following topics:

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.

Prerequisite software

Before you can use the WebSphere MQ Explorer, you must have the following installed on your Windows 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 (available from the Microsoft Web site at `http://www.microsoft.com`)

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

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 attempting to use the WebSphere MQ Explorer to manage WebSphere MQ on a server machine. Check that:

1. A command server is running for **any** queue manager being administered, started on the server by the STRMQMCSVR CL command.
2. A suitable TCP/IP listener exists for every remote queue manager. This is the WebSphere MQ listener started by the STRMQMLSR command.
3. The server connection channel, called SYSTEM.ADMIN.SVRCONN, exists on every remote queue manager. You ***must*** create this channel yourself. It is mandatory for every remote queue manager being administered. Without it, remote administration is not possible.

For further information on the WebSphere MQ Explorer, see the *WebSphere MQ System Administration Guide* supplied with your WebSphere MQ for Windows product.

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 PCFs, the MQAI, and also for remote administration.

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

There are separate control commands for starting and stopping the command server. You can perform the operations described in the following sections using the WebSphere MQ Services snap-in.

Starting the command server

To start the command server use this CL command:

```
STRMQMCSVR MQMNAME(saturn.queue.manager')
```

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

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 here `saturn.queue.manager`, the CL command is:

```
DSPMQMCSVR MQMNAME('saturn.queue.manager')
```

Issue this command on the target machine. If the command server is running, the panel shown in Figure 5 on page 38 appears:

```
               Display MQM Command Server (DSPMQMCSVR)


Queue manager name . . . . . . . > saturn.queue.manager

MQM Command Server Status. . . . > RUNNING









F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 5. Display MQM Command Server panel

Stopping a command server

To end a command server, the command, using the previous example is:

```
ENDMQMCSVR MQMNAME('saturn.queue.manager')
```

You can stop the command server in two different ways:

- For a controlled stop, use the ENDMQMCSVR command with the *CNTRLD option, which is the default.
- For an immediate stop, use the ENDMQMCSVR command with the *IMMED option.

Note: Stopping a queue manager also ends the command server associated with it (if one has been started).

Instrumentation events

You can use WebSphere MQ instrumentation events to monitor the operation of queue managers. See *WebSphere MQ Event Monitoring* for information about WebSphere MQ instrumentation events and how to use them.

Chapter 4. Work management

This chapter describes the way in which WebSphere MQ handles work requests, and details the options available for prioritizing and controlling the jobs associated with WebSphere MQ.

Warning

Do **not** alter WebSphere MQ work management objects unless you fully understand the concepts of OS/400 and WebSphere MQ work management. Additional information regarding subsystems and job descriptions can be found in *OS/400 Work Management*. Pay particular attention to the sections on "Job Starting and Routing" and "Batch Jobs".

WebSphere MQ for iSeries incorporates the OS/400 UNIX® environment and OS/400 threads. Do **not** make any changes to the objects in the Integrated File System (IFS).

During normal operations, a WebSphere MQ queue manager starts a number of batch jobs to perform different tasks. By default these batch jobs run in the QMQM subsystem that is created when WebSphere MQ is installed.

Work management refers to the process of tailoring WebSphere MQ tasks to obtain the optimum performance from your system, or to make administration simpler.

For example, you can:

- Change the run-priority of jobs to make one queue manager more responsive than another.
- Redirect the output of a number of jobs to a particular output queue.
- Make all jobs of a certain type run in a specific subsystem.
- Isolate errors to a subsystem.

Work management is carried out by creating or changing the job descriptions associated with the WebSphere MQ jobs. You can configure work management for:

- An entire WebSphere MQ installation
- Individual queue managers
- Individual jobs for individual queue managers

Description of WebSphere MQ tasks

When a queue manager is running, you see some or all of the following batch jobs running under the QMQM user profile in the WebSphere MQ subsystem. The jobs are described briefly in Table 1, to help you decide how to prioritize them.

Table 1. WebSphere MQ tasks.

Job name	Function
AMQALMPX	The checkpoint processor that periodically takes journal checkpoints
AMQCLMAA	Non-threaded TCP/IP listener
AMQCRSTA	TCP/IP-invoked channel responder

Description of tasks

Table 1. WebSphere MQ tasks. (continued)

Job name	Function
AMQCRS6B	LU62 receiver channel and client connection (see note).
AMQPCSEA	PCF command processor that handles PCF and remote administration requests
AMQRMPPA	Channel process pooling job
AMQRRMFA	Repository manager for clusters
AMQZDMAA	Deferred message handler
AMQZFUMA	Object authority manager (OAM)
AMQZLAA0	Queue manager agents that perform the bulk of the work for applications that connect to the queue manager using MQCNO_STANDARD_BINDING
AMQZXMA0	The execution controller that is the first job started by the queue manager. It deals with MQCONN requests, and starts agent processes to process WebSphere MQ API calls
RUNMQCHI	The channel initiator
RUNMQCHL	Sender channel job that is started for each sender channel
RUNMQDLQ	Dead letter queue handler
RUNMQLSR	Threaded TCP/IP listener
RUNMQTRM	Trigger monitor
Note: The LU62 receiver job runs in the communications subsystem and takes its run-time properties from the routing and communications entries that are used to start the job. See *WebSphere MQ Intercommunication* for more details.	

You can view all jobs connected to a queue manager, except listeners (which do not connect), using option 22 on the Work with Queue Manager (WRKMQM) panel. You can view listeners using the WRKMQMLSR command.

WebSphere MQ work management objects

When WebSphere MQ is installed, various objects are supplied in the QMQM library to assist with work management. These objects are the ones necessary for WebSphere MQ jobs to run in their own subsystem.

Sample job descriptions are provided for two of the WebSphere MQ batch jobs. If no specific job description is provided for a WebSphere MQ job, it runs with the default job description QMQMJOBD.

The work management objects that are supplied when you install WebSphere MQ are listed in Table 2 and the objects created for a queue manager are listed in Table 3 on page 41

Note: The work management objects can be found in the QMQM library and the queue manager objects can be found in the queue manager library.

Table 2. Work management objects

Name	Type	Description
AMQZLAA0	*JOBD	The job description that is used by the WebSphere MQ agent processes
AMQZXMA0	*JOBD	The job description that is used by WebSphere MQ execution controllers
QMQM	*SBSD	The subsystem in which all WebSphere MQ jobs run.

Table 2. Work management objects (continued)

Name	Type	Description
QMQM	*JOBQ	The job queue attached to the supplied subsystem
QMQMJOBD	*JOBD	The default WebSphere MQ job description, used if there is not a specific job description for a job
QMQMMSG	*MSGQ	The default message queue for WebSphere MQ jobs.
QMQMRUN20	*CLS	A class description for high priority WebSphere MQ jobs
QMQMRUN35	*CLS	A class description for medium priority WebSphere MQ jobs
QMQMRUN50	*CLS	A class description for low priority WebSphere MQ jobs

Table 3. Work management objects created for a queue manager

Name	Type	Description
AMQA000000	*JRNRCV	Local journal receiver
AMQAJRN	*JRN	Local journal
AMQAJRNMSG	*MSGQ	Local journal message queue
AMQCRC6B	*PGM	Program to start the LU6.2 connection
AMQRFCD4	*FILE	Queue manager channel definition file
QMQMMSG	*MSGQ	Queue manager message queue

How WebSphere MQ uses the work management objects

To understand how to configure work management, you must first understand how WebSphere MQ uses job descriptions.

The job description used to start the job controls many attributes of the job. For example:

- The job queue on which the job is queued and on which subsystem the job runs.
- The routing data used to start the job and class that the job uses for its run-time parameters.
- The output queue that the job uses for print files.

The process of starting a WebSphere MQ job can be considered in three steps:

1. WebSphere MQ selects a job description.

   WebSphere MQ uses the following technique to determine which job description to use for a batch job:

   a. Look in the queue manager library for a job description with the same name as the job. See "Understanding WebSphere MQ queue manager library names" on page 165 for further details about the queue manager library.

   b. Look in the queue manager library for the default job description QMQMJOBD.

   c. Look in the QMQM library for a job description with the same name as the job.

   d. Use the default job description, QMQMJOBD, in the QMQM library.

2. The job is submitted to the job queue.

Job descriptions supplied with WebSphere MQ have been set up, by default, to put jobs on to job queue QMQM in library QMQM. The QMQM job queue is attached to the supplied QMQM subsystem, so by default the jobs start running in the QMQM subsystem.

3. The job enters the subsystem and goes through the routing steps.

   When the job enters the subsystem, the routing data specified on the job description is used to find routing entries for the job.

   The routing data must match one of the routing entries defined in the QMQM subsystem, and this defines which of the supplied classes (QMQMRUN20, QMQMRUN35, or QMQMRUN50) is used by the job.

Note: If WebSphere MQ jobs do not appear to be starting, make sure that the subsystem is running and the job queue is not held,

If you have modified the WebSphere MQ work management objects, make sure everything is associated correctly. For example, if you specify a job queue other than QMQM/QMQM on the job description, make sure that an ADDJOBE is performed for the subsystem, that is, QMQM.

You can create a job description for each job documented in Table 1 on page 39 using the following worksheet as an example:

```
What is the queue manager library name? ___________
Does job description AMQZXMA0 exist in the queue manager library? Yes      No
Does job description QMQMJOBD exist in the queue manager library? Yes      No
Does job description AMQZXMA0 exist in the QMQM library?          Yes      No
Does job description QMQMJOBD exist in the QMQM library?          Yes      No
```

If you answer No to all these questions, create a global job description QMQMJOBD in the QMQM library.

The WebSphere MQ message queue

A WebSphere MQ message queue, QMQMMSG, is created in each queue manager library. Operating system messages are sent to this queue when queue manager jobs end and WebSphere MQ sends messages to the queue. For example, to report which journal receivers are needed at startup. Keep the number of messages in this message queue at a manageable size to make it easier to monitor.

Default system examples

The following examples show how an unmodified WebSphere MQ installation works when some of the standard jobs are submitted at queue manager startup time.

The first job that is started is the execution controller, AMQZXMA0.

1. Issue the **STRMQM** command for queue manager TESTQM.
2. WebSphere MQ searches the queue manager library QMTESTQM, firstly for job description AMQZXMA0, and then job description QMQMJOBD.

   Neither of these job descriptions exist, so WebSphere MQ looks for job description AMQZXMA0 in the product library QMQM. This job description exists, so it is used to submit the job.
3. The job description uses the WebSphere MQ default job queue, so the job is submitted to job queue QMQM/QMQM.
4. The routing data on the AMQZXMA0 job description is QMQMRUN20, so the system searches the subsystem routing entries for one that matches that data.

By default, the routing entry with sequence number 9900 has comparison data that matches QMQMRUN20, so the job is started with the class defined on that routing entry, which is also called QMQMRUN20.

5. The QMQM/QMQMRUN20 class has run priority set to 20, so the AMQZXMA0 job runs in subsystem QMQM with the same priority as most interactive jobs on the system.

The next job that starts is the checkpoint process, AMQALMPX.

1. WebSphere MQ searches the queue manager library QMTESTQM, firstly for job description AMQALPMX, and then job description QMQMJOBD.

   Neither of these job descriptions exist, so WebSphere MQ looks for job descriptions AMQALMPX and QMQMJOBD in the product library QMQM.

   Job description AMQALMPX does not exist but QMQMJOBD does, so QMQMJOBD is used to submit the job.

   Note: The QMQMJOBD job description is always used for WebSphere MQ jobs that do not have their own job description.

2. The job description uses the WebSphere MQ default job queue, so the job is submitted to job queue QMQM/QMQM.
3. The routing data on the QMQMJOBD job description is QMQMRUN35, so the system searches the subsystem routing entries for one that matches that data.

   By default, the routing entry with sequence number 9910 has comparison data that matches QMQMRUN35, so the job is started with the class defined on that routing entry, which is also called QMQMRUN35.

4. The QMQM/QMQMRUN35 class has run priority set to 35, so the AMQALMPX job runs in subsystem QMQM with a lower priority than most interactive jobs on the system, but higher priority than most batch jobs.

Configuring work management

The preceding examples show how WebSphere MQ job descriptions determine the run-time attributes of WebSphere MQ jobs.

The following examples show how you can change and create WebSphere MQ job descriptions to change the run-time attributes of WebSphere MQ jobs.

The key to the flexibility of WebSphere MQ work management lies in the two-tier way that WebSphere MQ searches for job descriptions:

- If you create or change job descriptions in a queue manager library, those changes override the global job descriptions in QMQM, but the changes are *local* and affect that particular queue manager alone.
- If you create or change global job descriptions in the QMQM library, those job descriptions affect all queue managers on the system, unless overridden locally for individual queue managers.

Configuration examples

1. The following example increases the priority of channel control jobs for an individual queue manager.

   To make the repository manager and channel initiator jobs, AMQRRMFA and RUNMQCHI, run as quickly as possible for queue manager TESTQM, carry out the following steps:

a. Create local duplicates of the QMQM/QMQMJOBD job description with the names of the WebSphere MQ processes that you want to control in the queue manager library. For example,

```
 CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
NEWOBJ(RUNMQCHI)
 CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
NEWOBJ(AMQRRMFA)
```

b. Change the routing data parameter on the job description to ensure that the jobs use the QMQMRUN20 class.

```
 CHGJOBD JOBD(QMTESTQM/RUNMQCHI) RTGDTA('QMQMRUN20')
 CHGJOBD JOBD(QMTESTQM/AMQRRMFA) RTGDTA('QMQMRUN20')
```

The AMQRRMFA and RUNMQCHI jobs for queue manager TESTQM now:

- Use the new local job descriptions in the queue manager library
- Run with priority 20, because the QMQMRUN20 class is used when the jobs enter the subsystem.

2. The following example defines a new run priority class for the QMQM subsystem.

   a. Create a duplicate class in the QMQM library, to allow other queue managers to access the class, by issuing the following command:

```
CRTDUPOBJ OBJ(QMQMRUN20) FROMLIB(QMQM) OBJTYPE(*CLS) TOLIB(QMQM)
       NEWOBJ(QMQMRUN10)
```

   b. Change the class to have the new run priority by issuing the following command:

```
CHGCLS CLS(QMQM/QMQMRUN10) RUNPTY(10)
```

   c. Add the new class definition to the subsystem by issuing the following command:

```
  ADDRTGE SBSD(QMQM/QMQM) SEQNBR(8999) CMPVAL('QMQMRUN10') PGM(QSYS/QCMD)
         CLS(QMQM/QMQMRUN10)
```

   Note: You can specify any numeric value for the routing sequence number, but the values must be in sequential order. This sequence number tells the subsystem the order in which routing entries are to be searched for a routing data match.

   d. Change the local or global job description to use the new priority class by issuing the following command:

```
 CHGJOBD JOBD(QMQMlibname/QMQMJOBD) RTGDTA('QMQMRUN10')
```

   Now all the queue manager jobs associated with the QMlibraryname use a run priority of 10.

3. The following example runs a queue manager in its own subsystem

   To make all the jobs for queue manager TESTQM run in the QBATCH subsystem, carry out the following steps:

   a. Create a local duplicate of the QMQM/QMQMJOBD job description in the queue manager library with the command

```
 CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM2)
```

   b. Change the job queue parameter on the job description to ensure that the jobs use the QBATCH job queue.

```
 CHGJOBD JOBD(QMTESTQM2/QMQMJOBD) JOBQ(*LIBL/QBATCH)
```

   Note: The job queue is associated with the subsystem description. If you find that the jobs are staying on the job queue, verify that the job

queue definition is defined on the SBSD. Use the DSPSBSD command for the subsystem and take option **6**, "Job queue entries".

All jobs for queue manager TESTQM2 now:

- Use the new local default job description in the queue manager library
- Are submitted to job queue QBATCH.

To ensure that jobs are routed and prioritized correctly:

- Either create routing entries for the WebSphere MQ jobs in subsystem QBATCH, or
- Rely on a catch-all routing entry that calls QCMD, irrespective of what routing data is used.

  This option works only if the maximum active jobs option for job queue QBATCH is set to *NOMAX. The system default is 1.

4. The following example creates another WebSphere MQ subsystem
   a. Create a duplicate subsystem in the QMQM library by issuing the following command:
      ```
      CRTDUPOBJ OBJ(QMQM) FROMLIB(QMQM) OBJTYPE(*SBSD) TOLIB(QMQM) NEWOBJ(QMQM2)
      ```
   b. Remove the QMQM job queue by issuing the following command:
      ```
      RMVJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM)
      ```
   c. Create a new job queue for the subsystem by issuing the following command:
      ```
      CRTJOBQ JOBQ(QMQM/QMQM2) TEXT('Job queue for MQSeries Queue Manager')
      ```
   d. Add a job queue entry to the subsystem by issuing the following command:
      ```
      ADDJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM2) MAXACT(*NOMAX)
      ```
   e. Create a duplicate QMQMJOBD in the queue manager library by issuing the following command:
      ```
      CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMlibraryname)
      ```
   f. Change the job description to use the new job queue by issuing the following command:
      ```
      CHGJOBD JOBD(QMlibraryname/QMQMJOBD) JOBQ(QMQM/QMQM2)
      ```
   g. Start the subsystem by issuing the following command:
      ```
      STRSBS SBSD(QMQM/QMQM2)
      ```

   Notes:

   a. You can specify the subsystem in any library. If for any reason the product is reinstalled, or the QMQM library is replaced, any changes you made are removed.
   b. All the queue manager jobs associated with the QMlibraryname now run under subsystem QMQM2.

5. The following example collects all output for a job type.

   To collect all the checkpoint process, AMQALMPX, job logs for multiple queue managers onto a single output queue, carry out the following steps:

   a. Create an output queue, for example
      ```
      CRTOUTQ OUTQ(MYLIB/CHCKPTLOGS)
      ```
   b. Create a global duplicate of the QMQM/QMQMJOBD job description, using the name of theWebSphere MQ process that you want to control, for example
      ```
      CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) NEWOBJ(AMQALMPX)
      ```

c. Change the output queue parameter on the job description to point to your new output queue, and change the job logging level so that all messages are written to the job log.

```
CHGJOBD JOBD(QMQM/AMQALMPX) OUTQ(MYLIB/CHKPTLOGS) LOG(4 00 *SECLVL)
```

All WebSphere MQ AMQALMPX jobs, for all queue managers, use the new global AMQALMPX job description, providing that there are no local overriding job descriptions in the local queue manager library.

All job log spool files for these jobs are now written to output queue CHKPTLOGS in library MYLIB.

Notes:

a. The preceding example works only if the QPJOBLOG, or any print file, has a value of *JOB for its output queue parameter. In the preceding example, the QSYS/QPDJOBLOG file needs OUTQ set to *JOB.

b. To change a system print file, use the CHGPRTF command. For example:

```
CHGPRTF PRTF(QJOBLOG) OUTQ(*JOB)
```

The *JOB option indicates that your job descriptions must be used.

c. You can send any spool files associated with the WebSphere MQ jobs to a particular output queue. However, verify that the print file being used has the appropriate value for the OUTQ parameter.

Chapter 5. Protecting WebSphere MQ objects

Security for WebSphere MQ for iSeries is implemented using the WebSphere MQ Object Authority Manager (OAM) in conjunction with OS/400 object level security.

Security considerations

You need to consider the following points when setting up authorities to the users in your enterprise:

1. Grant and revoke authorities to the WebSphere MQ for iSeries commands using the OS/400 **GRTOBJAUT** and **RVKOBJAUT** commands.
2. During installation of WebSphere MQ for iSeries the following special user profiles are created:

   QMQM
   Is used primarily for internal product-only functions. However, it can be used to run trusted applications using MQCNO_FASTPATH_BINDINGS; see the *WebSphere MQ Application Programming Guide* for further information.

   QMQMADM
   Is used as a group profile for administrators of WebSphere MQ. The group profile gives access to CL commands and WebSphere MQ resources.
3. If you are sending channel commands to remote queue managers, ensure that your user profile is a member of the group QMQMADM on the target system. For a list of PCF and MQSC channel commands, see "Channel command security" on page 65.
4. The group set associated with a user is cached when the group authorizations are computed by the OAM.

   Any changes made to a user's group memberships after the group set has been cached are not recognized until you restart the queue manager or execute RFRMQMAUT to refresh security.
5. Limit the number of users who have authority to work with commands that are particularly sensitive. These commands include:
   - Create Message Queue Manager (**CRTMQM**)
   - Delete Message Queue Manager (**DLTMQM**)
   - Start Message Queue Manager (**STRMQM**)
   - End Message Queue Manager (**ENDMQM**)
   - Start Command Server (**STRMQMCSVR**)
   - End Command Server (**ENDMQMCSVR**)
6. Channel definitions contain a security exit program specification. Channel creation and modification requires special considerations. Details of security exits is given in *WebSphere MQ Intercommunication*.
7. The channel exit and trigger monitor programs can be substituted. The security of such replacements is the responsibility of the programmer.

Understanding the Object Authority Manager

The OAM manages users' authorizations to manipulate WebSphere MQ objects, including queues and process definitions. It also provides a command interface through which you can grant or revoke access authority to an object for a specific group of users. The decision to allow access to a resource is made by the OAM, and the queue manager follows that decision. If the OAM cannot make a decision, the queue manager prevents access to that resource.

Resources you can protect with the OAM

Through the OAM you can control:

- Access to WebSphere MQ objects through the MQI. When an application program attempts to access an object, the OAM checks that the user profile making the request has the authorization for the operation requested.

  In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use PCF and MQSC commands.

Different groups of users can have different kinds of access authority to the same object. For example, for a specific queue, one group could perform both put and get operations; another group might be allowed only to browse the queue (MQGET with browse option). Similarly, some groups might have get and put authority to a queue, but not be allowed to alter or delete the queue.

WebSphere MQ for iSeries provides commands to grant, revoke, and display the authority that an application or user has to do the following:
- Issue WebSphere MQ for iSeries commands
- Perform operations on WebSphere MQ for iSeries objects

WebSphere MQ authorities

Access to WebSphere MQ objects is controlled by authorities to:
1. Issue the WebSphere MQ command
2. Access the WebSphere MQ objects referenced by the command

All WebSphere MQ for iSeries CL commands are shipped with an owner of QMQM, and the administration profile (QMQMADM) has *USE rights with the *PUBLIC access set to *EXCLUDE.

Changes to the authority structure of some of the product's CL commands allows public use of these commands, provided that you have the required OAM authority to the WebSphere MQ objects to make these changes.

Granting WebSphere MQ authorities to WebSphere MQ objects

WebSphere MQ for iSeries categorizes the product's CL commands into two groups:

Group 1

Users must be in the QMQMADM user group, or have *ALLOBJ authority, to process these commands. Users having either of these authorities can process all commands in all categories without requiring any extra authority.

Note: These authorities override any OAM authority.

These commands can be grouped as follows:

- Authentication Information Commands
  - CHGMQMAUTI, Change MQM Authentication Information
  - CPYMQMAUTI, Copy MQM Authentication Information
  - CRTMQMAUTI, Create MQM Authentication Information
  - DLTMQMAUTI, Delete MQM Authentication Information
- License Unit Command
  - CHGMQMCAP, Change License Units
- Channel Commands
  - CHGMQMCHL, Change MQM Channel
  - CPYMQMCHL, Copy MQM Channel
  - CRTMQMCHL, Create MQM Channel
  - DLTMQMCHL, Delete MQM Channel
  - RSVMQMCHL, Resolve MQM Channel
- Command Server Commands
  - ENDMQMCSVR, End MQM Command Server
  - STRMQMCSVR, Start MQM Command Server
- Dead-Letter Queue Handler Command
  - STRMQMDLQ, Start WebSphere MQ Dead-Letter Queue Handler
- Media Recovery Commands
  - RCDMQMIMG, Record MQM Object Image
  - RCRMQMOBJ, Recreate MQM Object
- Queue Manager Commands
  - CRTMQM, Create Message Queue Manager
  - DLTMQM, Delete Message Queue Manager
  - ENDMQM, End Message Queue Manager
  - STRMQM, Start Message Queue Manager
- Security Commands
  - GRTMQMAUT, Grant MQM Object Authority
  - RVKMQMAUT, Revoke MQM Object Authority
- Trace Commands
  - TRCMQM, Trace MQM Job
- Transaction Commands
  - STRMQMTRN, Start WebSphere MQ Transaction
  - WRKMQMTRN, Work with WebSphere MQ Transactions
  - RSVMQMTRN, Resolve WebSphere MQ Transaction
- Trigger Monitor Commands
  - STRMQMTRM, Start Trigger Monitor
- WebSphere MQ Commands
  - RUNMQSC, Run MQSC Commands
  - STRMQMMQSC, Start MQSC Commands

Group 2

The rest of the commands, for which two levels of authority are required:

1. OS/400 authority to run the command. A WebSphere MQ administrator sets this using the **GRTOBJAUT** command to override the *PUBLIC(*EXCLUDE) restriction for a user or group of users.

   For example:

   ```
   GRTOBJAUT OBJ(DSPMQMQ) OBJTYPE(*CMD) USER(MQUSER) AUT(*USE)
   ```

2. WebSphere MQ authority to manipulate the WebSphere MQ objects associated with the command, or commands, given the correct OS/400 authority in Step 1.

   This authority is controlled by the user having the appropriate OAM authority for the required action, set by a WebSphere MQ administrator using the **GRTMQMAUT** command

   For example:

   ```
   CHGMQMQ *connect authority to the queue manager + *admchg authority to
     the queue
   ```

The commands can be grouped as follows:

- Display commands

  To process the DSP commands you must grant the user `*connect` and `*admdsp` authority to the queue manager, together with any specific option listed:

  DSPMQMAUT, Display MQM Object Authority
  DSPMQMAUTI, Display MQM Authentication Information
  DSPMQMCAP, Display License Units
  DSPMQMCHL, Display MQM Channel
  DSPMQMCSVR, Display MQM Command Server
  DSPMQMNL, Display MQM Namelist – `*admdsp` to the namelist
  DSPMQMOBJN, Display MQM Object Names
  DSPMQMPRC, Display MQM Process – `*admdsp` to the process
  DSPMQM, Display Message Queue Manager
  DSPMQMQ, Display MQM Queue – `*admdsp` to the queue

- Work with commands

  To process the WRK commands and display the options panel you must grant the user `*connect` and `*admdsp` authority to the queue manager, together with any specific option listed:

  WRKMQMAUT, Work with MQM Object Authority
  WRKMQMAUTD, Work with MQM Object Authority Data
  WRKMQMAUTI, Work with MQM Authentication Information
  WRKMQMCHL, Work with MQM Channel
  WRKMQMCHST, Work with MQM Channel Status
  WRKMQMCL, Work with MQM Clusters
  WRKMQMCLQM, Work with MQM Cluster Queue Manager
  WRKMQM, Work with Message Queue Managers
  WRKMQMLSR, Work with MQM Listener
  WRKMQMMSG, Work with MQM Messages – `*browse` to the queue
  WRKMQMNL, Work with MQM Namelists
  WRKMQMPRC, Work with MQM Processes
  WRKMQMQ, Work with MQM queues
  WRKMQMQSTS, Work with MQM Queue Status

- Other Channel commands

  To process the channel commands you must grant the user the specific authorities listed:

  – ENDMQMCHL, End MQM Channel

    This requires `*connect` authority to the queue manager and `*allmqi` authority to the transmission queue associated with the channel.

  – ENDMQMLSR, End MQM Listener

    This requires no WebSphere MQ object authority.

  – PNGMQMCHL, Ping MQM Channel

This requires *connect* and *inq*authority to the queue manager.

– RSTMQMCHL, Reset MQM Channel

This requires *connect* authority to the queue manager.

– STRMQMCHL, Start MQM Channel

This requires *connect* authority to the queue manager and *allmqi* authority to the transmission queue associated with the channel.

– STRMQMCHLI, Start MQM Channel Initiator

This requires *connect* and *inq*authority to the queue manager, and *allmqi* authority to the initiation queue associated with the transmission queue of the channel.

– STRMQMLSR, Start MQM Listener

This requires no WebSphere MQ object authority.

- Other commands:

  CCTMQM, Connect to Message Queue Manager
  CHGMQM, Change Message Queue Manager
  CHGMQMNL, Change MQM Namelist
  CHGMQMPRC, Change MQM Process
  CHGMQMQ, Change MQM Queue
  CLRMQMQ, Clear MQM Queue
  CPYMQMNL, Copy MQM Namelist
  CPYMQMPRC, Copy MQM Process
  CPYMQMQ, Copy MQM Queue
  CRTMQMNL, Create MQM Namelist
  CRTMQMPRC, Create MQM Process
  CRTMQMQ, Create MQM Queue
  CVTMQMDTA, Convert MQM Data Type Command
  DLTMQMNL, Delete MQM Namelist
  DLTMQMPRC, Delete MQM Process
  DLTMQMQ, Delete MQM Queue
  DSCMQM, Disconnect from Message Queue Manager
  RFRMQMAUT, Refresh Security
  RFRMQMCL, Refresh Cluster
  RSMMQMCLQM, Resume Cluster Queue Manager
  RSTMQMCL, Reset Cluster
  SPDMQMCLQM, Suspend Cluster Queue Manager

Access authorizations

Authorizations defined by the AUT keyword on the **GRTMQMAUT** and **RVKMQMAUT** commands can be categorized as follows:

- Authorizations related to MQI calls
- Authorization-related administration commands
- Context authorizations
- General authorizations, that is, for MQI calls, for commands, or both

The following tables list the different authorities, using the AUT parameter for MQI calls, Context calls, MQSC and PCF commands, and generic operations.

Table 4. Authorizations for MQI calls

AUT	Description
***ALTUSR**	Allow another user's authority to be used for MQOPEN and MQPUT1 calls.

Table 4. Authorizations for MQI calls (continued)

AUT	Description
***BROWSE**	Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
***CONNECT**	Connect the application to the specified queue manager by issuing an MQCONN call.
***GET**	Retrieve a message from a queue by issuing an MQGET call.
***INQ**	Make an inquiry on a specific queue by issuing an MQINQ call.
***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. If you open a queue for multiple options, you must be authorized for each of them.

Table 5. Authorizations for context calls

AUT	Description
***PASSALL**	Pass all context on the specified queue. All the context fields are copied from the original request.
***PASSID**	Pass identity context on the specified queue. The identity context is the same as that of the request.
***SETALL**	Set all context on the specified queue. This is used by special system utilities.
***SETID**	Set identity context on the specified queue. This is used by special system utilities.

Table 6. Authorizations for MQSC and PCF calls

AUT	Description
***ADMCHG**	Change the attributes of the specified object.
***ADMCLR**	Clear the specified queue (PCF Clear queue command only).
***ADMCRT**	Create objects of the specified type.
***ADMDLT**	Delete the specified object.
***ADMDSP**	Display the attributes of the specified object.

Table 7. Authorizations for generic operations

AUT	Description
***ALL**	Use all operations applicable to the object.
***ALLADM**	Perform all administration operations applicable to the object.
***ALLMQI**	Use all MQI calls applicable to the object.

Using the GRTMQMAUT command

Provided that you have the required authorization, you can use the **GRTMQMAUT** command to grant authorization of a user profile or user group to access a particular object. The following examples illustrate how the **GRTMQMAUT** command is used:

1.
```
GRTMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
          AUT(*BROWSE *PUT) MQMNAME('saturn.queue.manager')
```

   In this example:

- RED.LOCAL.QUEUE is the object name.
- *LCLQ (local queue) is the object type.
- GROUPA is the name of a user profile on the system whose authorizations are to change. This can be used as a group profile for other users.
- *BROWSE and *PUT are the authorizations being granted to the specified queue.

  *BROWSE adds authorization to browse messages on the queue (to issue MQGET with the browse option).

  *PUT adds authorization to put (MQPUT) messages on the queue.
- saturn.queue.manager is the queue manager name.

2. The following command grants to users JACK and JILL all applicable authorizations, to all process definitions, for the default queue manager.

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*PRC) USER(JACK JILL) AUT(*ALL)
```

3. The following command grants user GEORGE authority to put a message on the queue ORDERS, on the queue manager TRENT.

```
GRTMQMAUT OBJ(TRENT) OBJTYPE(*MQM) USER(GEORGE) AUT(*CONNECT) MQMNAME (TRENT)
GRTMQMAUT OBJ(ORDERS) OBJTYPE(*Q) USER(GEORGE) AUT(*PUT) MQMNAME (TRENT)
```

Using the RVKMQMAUT command

Provided that you have the required authorization, you can use the **RVKMQMAUT** command to remove previously granted authorization of a user profile or user group to access a particular object. The following examples illustrate how the **RVKMQMAUT** command is used:

1.
```
RVKMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
AUT(*PUT) MQMNAME('saturn.queue.manager')
```

   The authority to put messages to the specified queue, that was granted in the previous example, is removed for GROUPA.

2.
```
RVKMQMAUT OBJ(PAY*) OBJTYPE(*Q) USER(*PUBLIC) AUT(*GET) +
MQMNAME(PAYROLLQM)
```

   Authority to get messages from any queue whose name starts with the characters PAY, owned by queue manager PAYROLLQM, is removed from all users of the system unless they, or a group to which they belong, have been separately authorized.

Using the DSPMQMAUT command

The display MQM authority (**DSPMQMAUT**) command shows, for the specified object and user, the list of authorizations that the user has for the object. The following example illustrates how the command is used:

```
DSPMQMAUT OBJ(ADMINNL) OBJTYPE(*NMLIST) USER(JOE) OUTPUT(*PRINT) +
MQMNAME(ADMINQM)
```

Using the RFRMQMAUT command

The refresh MQM security (**RFRMQMAUT**) command enables you to update the OAM's authorization group information immediately, reflecting changes made at the operating system level, without needing to stop and restart the queue manager. The following example illustrates how the command is used:

```
RFRMQMAUT MQMNAME(ADMINQM)
```

Understanding the authorization specification tables

The authorization specification tables starting on page 54 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 **GRTMQMAUT** and **RVKMQMAUT** commands for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword `*BROWSE`; similarly, the keyword MQZAO_SET_ALL_CONTEXT corresponds to the keyword *SETALL and so on. These constants are defined in the header file cmqzc.h, which is supplied with the product.

MQI authorizations

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls require authorization checks: MQCONN, MQOPEN, MQPUT1, and MQCLOSE.

For MQOPEN and MQPUT1, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application can be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of name resolution that is not a queue-manager alias, unless the queue-manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the particular object being opened; in some cases additional queue-independent authority, obtained through an authorization for the queue-manager object, is required.

Table 8, Table 9 on page 55, Table 10 on page 55, and Table 11 on page 55 summarize the authorizations needed for each call.

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.

Table 8. Security authorization needed for MQCONN calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT

Table 9. Security authorization needed for MQOPEN calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQOO_INQUIRE	MQZAO_INQUIRE **(2)**	MQZAO_INQUIRE **(2)**	MQZAO_INQUIRE **(2)**
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(3)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(4)**	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(5)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT **(5, 6)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT **(5, 6)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7)**
MQOO_SET_ ALL_CONTEXT **(5, 8)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
MQOO_OUTPUT (Transmission queue) **(9)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	**(10)**	**(10)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(10, 11)**

Table 10. 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 **(12)**	Not applicable	No check
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT **(12)**	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT **(12)**	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7)**
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT **(12)**	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
(Transmission queue) **(9)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
MQPMO_ALTERNATE_ USER_AUTHORITY	**(13)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(11)**

Table 11. Security authorization needed for MQCLOSE calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQCO_DELETE	MQZAO_DELETE **(14)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(14)**	Not applicable	Not applicable

Notes for the tables:

1. If a model queue is being opened:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.

2. Either the queue, process, namelist, or queue manager object is checked, depending on the type of object being opened.
3. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
4. This check is performed for all output cases, except the case specified in note 9.
5. MQOO_OUTPUT must also be specified.
6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
7. This authority is required for both the queue manager object and the particular queue.
8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
9. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
11. This authorization allows any *AlternateUserId* to be specified.
12. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
13. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
14. The check is carried out only if both of the following are true:
    - A permanent dynamic queue is being closed and deleted.
    - The queue was not created by the MQOPEN that returned the object handle being used.

    Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all the following that are relevant to the object type:
   - MQZAO_CONNECT
   - MQZAO_INQUIRE
   - MQZAO_SET
   - MQZAO_BROWSE
   - MQZAO_INPUT
   - MQZAO_OUTPUT
   - MQZAO_PASS_IDENTITY_CONTEXT
   - MQZAO_PASS_ALL_CONTEXT
   - MQZAO_SET_IDENTITY_CONTEXT
   - MQZAO_SET_ALL_CONTEXT
   - MQZAO_ALTERNATE_USER_AUTHORITY

2. MQZAO_DELETE (see note 14) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. *No check* means that no authorization checking is carried out.
4. *Not applicable* means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Administration authorizations

These authorizations allow a user to issue administration commands. This can be an MQSC command as an escape PCF message or as a PCF command itself. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

Authorizations for MQSC commands in escape PCFs

Table 12 summarizes the authorizations needed for each MQSC command that is contained in an escape PCF.

Table 12. MQSC commands and security authorization needed

(2) Authorization required for:	Queue object	Process object	Queue manager object	Namelists	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 **(3)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**
DEFINE object REPLACE **(3, 5)**	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 12:

1. The user identifier, under which the program that submits the command is running, must also have MQZAO_CONNECT authority to the queue manager.
2. Either the queue, process, namelist, or queue manager object is checked, depending on the type of object.
3. 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.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the **GRTMQMAUT** command.
5. This applies if the object to be replaced already exists. If it does not, the check is as for DEFINE object NOREPLACE.

General notes:

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The authority to execute an escape PCF depends on the MQSC command within the text of the escape PCF message.
3. *Not applicable* means that authorization checking is not relevant to this operation. For example, you cannot issue a CLEAR QLOCAL on a queue manager object.

Authorizations for PCF commands

Table 13 summarizes the authorizations needed for each PCF command.

Table 13. PCF commands and security authorization needed

(2) Authorization required for:	**Queue object**	**Process object**	**Queue manager object**	**Namelists**	**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) **(3)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE (4)	MQZAO_CREATE (4)
Copy object (with replace) **(3, 6)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE	MQZAO_CHANGE
Create object (without replace) **(5)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**
Create object (with replace) **(5, 6)**	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 13:

1. The user identifier under which the program submitting the command is running must also have authority to connect to its local queue manager, and to open the command administration queue for output.
2. Either the queue, process, namelist, or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the **GRTMQMAUT** command.
5. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
6. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

General notes:

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The special authorization MQZAO_ALL_ADMIN includes all the following that are relevant to the object type:
   - MQZAO_CHANGE
   - MQZAO_CLEAR
   - MQZAO_DELETE
   - MQZAO_DISPLAY

   MQZAO_CREATE is not included because it is not specific to a particular object or object type.
3. *No check* means that no authorization checking is carried out.

4. *Not applicable* means that authorization checking is not relevant to this operation. For example, you cannot use a Clear Queue command on a process object.

Generic OAM profiles

OAM generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate GRTMQMAUT commands against each individual object when it is created. Using generic profiles in the GRTMQMAUT command enables you to set a generic authority for all future objects created that fit that profile.

The rest of this section describes the use of generic profiles in more detail:
- "Using wildcard characters"
- "Profile priorities"

Using wildcard characters

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the ? wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects created with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

? Use the question mark (?) instead of any single character. For example, `AB.?D` would apply to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

* Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` would apply to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it would **not** apply to `ABC.JKL`; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` would apply to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

** Use the double asterisk (**) ***once*** in a profile name as:

- The entire profile name to match all object names. For example, if you use the keyword `OBJTYPE (*PRC)` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
GRTMQMAUT OBJ(AB.*) OBJTYPE(*Q) USER(FRED) AUT(*PUT) MQMNAME(MYQMGR)
GRTMQMAUT OBJ(AB.C*) OBJTYPE(*Q) USER(FRED) AUT(*GET) MQMNAME(MYQMGR)
```

Generic profiles

The first gives put authority to all queues for the principal FRED with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either GRTMQMAUT could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific 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. **

Specifying the installed authorization service

The parameter Service Component name on GRTMQMAUT and RVKMQMAUT allows you to specify the name of the installed authorization service component.

Selecting F24 on the initial panel, followed by F9=All parameters on the next panel of either command, allows you to specify either the installed authorization component (*DFT) or the name of the required authorization service component specified in the Service stanza of the queue manager's qm.ini file.

DSPMQMAUT also has this extra parameter. This allows you to search all the installed authorization components (*DFT), or the specified authorization-service component name, for the specified object name, object type, and user

Working without authority profiles

You can work with authority profiles, as explained in "Working with authority profiles", or without them, as explained here.

To use no authority profiles, use *NONE as an Authority parameter on GRTMQMAUT to create profiles without authority. This leaves any existing profiles unchanged.

On RVKMQMAUT, use *REMOVE as an Authority parameter to remove an existing authority profile.

Working with authority profiles

There are two commands associated with authority profiling:

- WRKMQMAUT
- WRKMQMAUTD

You can access these commands directly from the command line, or from the WRKMQM panel by:

1. Typing in the queue manager name and pressing the Enter key to access the WRKMQM results panel

2. Selecting F23=More options on this panel

Option 24 selects the results panel for the WRKMQMAUT command (see Figure 7 on page 62) and option 25 selects the WRKMQMAUTI command, which is used with the SSL bindings layer.

WRKMQMAUT

This command allows you to work with the authority data held in the authority queue. Figure 6 shows the input panel for this command.

Note: To run this command you must have *connect and *admdsp authority to the queue manager. However, to create or delete a profile, you need QMQMADM authority.

If you output the information to the screen, a list of authority profile names, together with their types, is displayed. If you print the output, you receive a detailed list of all the authority data, the registered users, and their authorities.

```
                    Work with MQ Authority (WRKMQMAUT)

Type choices, press Enter.

Object/Profile name  . . . . . .   *ALL

Object type  . . . . . . . . . .   *ALL          *Q, *PRC, *MQM, *NMLIST...
Output . . . . . . . . . . . . .   *             *, *PRINT
Message Queue Manager name . . .   *DFT

                                                                      Bottom

===> ____________________________________________________________________________
F3=Exit   F4=Prompt    F5=Refresh    F12=Cancel     F13= How to use this display
F24=More keys
```

Figure 6. Work with MQM Authority panel – input display

Entering an object or profile name on this panel, and pressing ENTER takes you to the results panel for WRKMQMAUT, shown in Figure 7 on page 62.

```
                                Work with MQ Authority

Queue Manager Name . . :   *DFT

Type options, press Enter.
  4=Delete   12=Work with profile

Opt     Authority Profile Name                              Type
        SYSTEM.DEFAULT.NAMELIST                             *NMLIST
        SYSTEM.DEFAULT.PROCESS                              *PRC
        SYSTEM.DEFAULT.REMOTE.QUEUE                         *Q
        SYSTEM.MQSC.REPLY.QUEUE                             *Q
        SYSTEM.PENDING.DATA.QUEUE                           *Q

                                                                        Bottom
===>
F3=Exit   F4=Prompt      F5=Refresh   F6=Create   F9=Retrieve   F12=Cancel
F16=Repeat position to   F17=Position to          F21=Print
```

Figure 7. Work with MQM Authority panel – results display

If you select `4=Delete`, you go to a new panel from which you can confirm that you want to delete all the user names registered to the generic authority profile name you specify. This option runs RVKMQMAUT with the option *REMOVE for all the users, and applies **only** to generic profile names.

If you select `12=Work with profile` you go to the WRKMQMAUTD command results panel, as explained in "WRKMQMAUTD".

WRKMQMAUTD

This command allows you to display all the users registered with a particular authority profile name and object type. To run this command you must have `*connect` and `*admdsp` authority to the queue manager. However, to grant, run, create, or delete a profile you need QMQMADM authority.

Figure 8 on page 63 shows the input panel of the WRKMQMAUTD command.

```
                  Work with MQ Authority Data (WRKMQMAUTD)

Type choices, press Enter.

Object/Profile name  . . . . . .

Object type  . . . . . .                *Q, *PRC, *MQM, *NMLIST...
User name  . . . . . .                  Name, *PUBLIC, *ALL
Message Queue Manager name . . .

                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 8. Work with MQM Authority Data input panel

Selecting F24=More keys from the initial input panel, followed by option F9=All Parameters displays the Service Component Name as for GRTMQMAUT and RVKMQMAUT.

Figure 9 shows the results panel of the WRKMQMAUTD command.

```
                        Work with MQ Authority Data

Queue Manager Name . . :   *DFT
Authority Profile Name :   SYSTEM.DEFAULT.PROCESS
Object Type  . . . . . :   *PRC

Type options, press Enter.
  2=Grant   3=Revoke   4=Delete   5=Display


Opt  UserName    GET  BROWSE  PUT  CONNECT  INQ  SET  ALTUSR
     MQUSER       X            X             X
     QMQMADM      X     X      X      X      X    X     X









                                                                      Bottom
===>
F3=Exit   F4=Prompt   F5=Refresh    F6=Create    F9=Retrieve
F11=Display Object Authorizations   F12=Cancel   F24=More keys
```

Figure 9. Work with MQM Authority Data output panel

Note: The F11=Display Object Authorizations key toggles between the following types of authorities:

- Object authorizations
- Context authorizations
- MQI authorizations

The options on the screen are:

2=Grant	Takes you to the GRTMQMAUT panel to add to the current authorities.
3=Revoke	Takes you to the RVKMQMAUT panel to remove some of the current definitions
4=Delete	Takes you to a panel that allows you to delete the authority data for specified users. This runs RVKMQMAUT with the option *REMOVE.
5=Display	Takes you to the existing DSPMQMAUT command
F6=Create	Takes you to the GRTMQMAUT panel that allows you to create a new profile authority record.

Object Authority Manager guidelines

Some operations are particularly sensitive; limit them to privileged users. For example,

- Accessing some special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE
- Running programs that use full MQI context options
- Creating and copying application queues

Queue manager directories

The directories and libraries containing queues and other queue manager data are private to the product. Do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Queues

The authority to a dynamic queue is based on, but is not necessarily the same as, that of the model queue from which it is derived.

For alias queues and remote queues, the authorization is that of the object itself, not the queue to which the alias or remote queue resolves. It is possible to authorize a user profile to access an alias queue that resolves to a local queue to which the user profile has no access permissions.

Limit the authority to create queues to privileged users. If you do not, users can bypass the normal access control simply by creating an alias.

Alternate-user authority

Alternate-user authority controls whether one user profile can use the authority of another user profile when accessing a WebSphere MQ object. This is essential where a server receives requests from a program and the server wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example:

- A server program running under user profile PAYSERV retrieves a request message from a queue that was put on the queue by user profile USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user profile (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user profile, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user profile when it opens the reply-to queue.

The alternate-user profile is specified on the *AlternateUserId* field of the object descriptor.

Note: You can use alternate-user profiles on any WebSphere MQ object. Use of an alternate-user profile does not affect the user profile used by any other resource managers.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message.

For descriptions of the message descriptor fields relating to context, see the *WebSphere MQ Application Programming Reference*.

For information about the context options, see the *WebSphere MQ Application Programming Guide*.

Remote security considerations

For remote security, consider:

Put authority
For security across queue managers you can specify the put authority that is used when a channel receives a message sent from another queue manager.

Specify the channel attribute PUTAUT as follows:

DEF Default user profile. This is the QMQM user profile under which the message channel agent is running.

CTX The user profile in the message context.

Transmission queues
Queue managers automatically put remote messages on a transmission queue; no special authority is required for this. However, putting a message directly on a transmission queue requires special authorization.

Channel exits
Channel exits can be used for added security.

For more information about remote security, see *WebSphere MQ Intercommunication*.

Channel command security

Channel commands can be issued as PCF commands, through the MQAI, MQSC commands, and control commands.

PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote WebSphere MQ system. The user profile, as specified in the message descriptor of the PCF message, must have the appropriate authorizations in the relevant group on the target system.

On WebSphere MQ for iSeries the actual group is QMQMADM, and on UNIX systems the name of the group is mqm.

These commands are:
- *ChangeChannel*
- *CopyChannel*
- *CreateChannel*
- *DeleteChannel*
- *PingChannel*
- *ResetChannel*
- *StartChannel*
- *StartChannelInitiator*
- *StartChannelListener*
- *StopChannel*
- *ResolveChannel*

See *WebSphere MQ Programmable Command Formats and Administration Interface* for the PCF security requirements.

MQSC channel commands

You can issue MQSC channel commands to a remote WebSphere MQ system either by sending the command directly in a PCF escape message or by issuing the command using **STRMQMMQSC**. The user profile as specified in the message descriptor of the associated PCF message must belong to the relevant group on the target system. (PCF commands are implicit in MQSC commands issued from **STRMQMMQSC**) These commands are:
- ALTER CHANNEL
- DEFINE CHANNEL
- DELETE CHANNEL
- PING CHANNEL
- RESET CHANNEL
- START CHANNEL
- START CHINIT
- START LISTENER
- STOP CHANNEL
- RESOLVE CHANNEL

For MQSC commands issued from the **STRMQMMQSC** command, the user profile in the PCF message is normally that of the current user.

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 CL and MQSC commands and queue manager and channel parameters that define the SSL support required in detail.

The following CL commands support SSL:

WRKMQMAUTI
Work with the attributes of an authentication information object.

CHGMQMAUTI
Modify the attributes of an authentication information object.

CRTMQMAUTI
Create a new authentication information object.

CPYMQMAUTI
Create a new authentication information object by copying an existing one.

DLTMQMAUTI
Delete an authentication information object.

DSPMQMAUTI
Displays the attributes for a specific authentication information object.

For an overview of channel security using SSL, see *WebSphere MQ Security*.

For details of PCF commands associated with SSL, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

Chapter 6. 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.

Note: It is often preferable to avoid placing messages on a DLQ. For information about the use and avoidance of DLQs, see the *WebSphere MQ Application Programming Guide*.

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

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

Invoking the DLQ handler

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

- As parameters to STRMQMDLQ from the command prompt. For example:

```
STRMQMDLQ UDLMSGQ(ABC1.DEAD.LETTER.QUEUE) SRCMBR(QRULE) SRCFILE(library/QTXTSRC)
          MQMNAME(MY.QUEUE.MANAGER)
```

- In the rules table. For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE)
```

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

The above examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the default queue manager.

If you do not specify the DLQ or the queue manager as shown above, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The STRMQMDLQ command takes its input from the rules table.

You must be authorized to access both the DLQ itself, and any message queues to which messages on the DLQ are forwarded, in order to run the DLQ handler. You must also be authorized to assume the identity of other users, for the DLQ to put messages on queues with the authority of the user ID in the message context.

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. You can specify only one of these.
- All keywords are optional.

INPUTQ (*QueueName***|' ')**

The name of the DLQ you want to process:

1. Any UDLMSGQ value (or *DFT) you specify as a parameter to the STRMQMDLQ command overrides any INPUTQ value in the rules table.
2. If you specify a blank UDLMSGQ value as a parameter to the STRMQMDLQ command, the INPUTQ value in the rules table is used.
3. If you specify a blank UDLMSGQ value as a parameter to the STRMQMDLQ command, and a blank INPUTQ value in the rules table, the system default dead-letter queue is used.

INPUTQM (*QueueManagerName***|' ')**

The name of the queue manager that owns the DLQ named on the INPUTQ keyword.

If you do not specify a queue manager, or you specify INPUTQM(' ') in the rules table, the system uses the default queue manager for the installation.

RETRYINT (*Interval***|60)**

The interval, in seconds, at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (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 Causes the DLQ handler to wait indefinitely.

NO Causes the DLQ handler to terminate when it detects that the DLQ is either empty or contains no messages that it can process.

nnn Causes the DLQ handler to wait for *nnn* seconds for new work to arrive before terminating, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, re-invoke it using triggering.

You can supply the name of the DLQ as an input parameter to the STRMQMDLQ command, as an alternative to including control data in the rules table. If any value is specified both in the rules table and on input to the STRMQMDLQ command, the value specified on the STRMQMDLQ command takes precedence.

Note: If a control-data entry is included in the rules table, it ***must*** be the first entry in the table.

Rules (patterns and actions)

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
 ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make 3 attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

This section describes the keywords that you can include in a rule. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives. You can specify only one of these.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched). It then describes the action keywords (those that determine how the DLQ handler is to process a matching message).

The pattern-matching keywords

The pattern-matching keywords are described below. Use them to specify values against which messages on the DLQ are matched. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData***|*)**
The *ApplIdentityData* value of the message on the DLQ, specified in the message descriptor, MQMD.

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 queue manager name for the message queue for which the message is destined.

FEEDBACK (*Feedback***|*)**
When the *MsgType* value is MQMT_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues.

FORMAT (*Format* **| *)**
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType* **| *)**
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

PERSIST (*Persistence* **| *)**
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

REASON (*ReasonCode* **| *)**
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName* **| *)**
The reply-to queue name specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName* **| *)**
The queue manager name of the reply-to queue specified in the REPLYQ keyword.

USERID (*UserIdentifier* **| *)**
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD.

The action keywords

The action keywords are described below. Use them to describe how a matching message is processed.

ACTION (DISCARD | IGNORE | RETRY | FWD)
The action taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Causes the message to be deleted from the DLQ.

IGNORE
Causes the message to be left on the DLQ.

RETRY
Causes the DLQ handler to try again to put the message on its destination queue.

FWD Causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The RETRYINT keyword of the control data controls the interval between attempts.

FWDQ (*QueueName***|&DESTQ|&REPLYQ)**
The name of the message queue to which the message is forwarded when you select the ACTION keyword.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

You can specify REPLYQ (?*) in the message pattern to avoid error messages, when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field.

FWDQM (*QueueManagerName***|&DESTQM|&REPLYQM|' ')**
The queue manager of the queue to which a message is forwarded.

QueueManagerName
The queue manager name for the queue to which the message is forwarded when you select the ACTION (FWD) keyword.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' ' FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (YES|NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF|CTX)
The authority with which messages should be put by the DLQ handler:

DEF Puts messages with the authority of the DLQ handler itself.

CTX Causes the messages to be put with the authority of the user ID in the message context. You must be authorized to assume the identity of other users, if you specify PUTAUT (CTX).

RETRY (*RetryCount***|1)**
The number of times, in the range 1–999 999 999, to attempt an action (at the interval specified on the RETRYINT keyword of the control data).

Note: The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If you restart the DLQ handler, the count of attempts made to apply a rule is reset to zero.

Rules table conventions

The rules table must adhere to the following conventions regarding its syntax, structure, and contents:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last non-blank character on a line to indicate that the rule continues from the first non-blank character in the next line. Use the minus sign (–) as the last non-blank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

  ```
  APPLNAME('ABC+
    D')
  ```

  results in 'ABCD'.

  ```
  APPLNAME('ABC-
    D')
  ```

  results in 'ABC D'.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in 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 (?) in place of any single character, except a trailing blank. You can use the asterisk (*) in

place of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.

- You cannot include wildcard characters in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
- Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
- Numeric parameters cannot include the question mark (?) wildcard character. You can include the asterisk (*) in place of an entire numeric parameter, but the asterisk cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

```
MSGTYPE(2)          Only reply messages are eligible
MSGTYPE(*)          Any message type is eligible
MSGTYPE('*')        Any message type is eligible
```

  However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.
- Numeric parameters must be in the range 0–999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8-character field:

```
'ABCDEFGH'          8 characters
'A*C*E*G*I'         5 characters excluding asterisks
'*A*C*E*G*I*K*M*O*' 8 characters excluding asterisks
```

- Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

Processing the rules table

The DLQ handler searches the rules table for a rule 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 a rule with a matching pattern is found, the rules table attempts the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its

RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

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 default, so that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors flagged at that time. (Error messages issued by the DLQ handler are described in *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 is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized, because the validation of the rules table is designed to eliminate the generation of repetitive errors.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still keeps a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ will be seen, even if the DLQ is defined as first-in first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, try to ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which 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 causes messages that fall through to the final rule in the table to be forwarded to the queue REALLY.DEAD.QUEUE, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table

Here is an example rules table that contains a single control-data entry and several rules:

```
*********************************************************************
*        An example rules table for the STRMQMDLQ command           *
*********************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* STRMQMDLQ, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to STRMQMDLQ,
* use the DLQ defined for the local queue manager.
*
inputqm(' ')  inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.

* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation is always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never does things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
  action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)
```

Example rules table

```
* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it must be able
* to cope with the message being lost, so we can afford to
* discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We do not have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Chapter 7. Backup, recovery, and restart

WebSphere MQ for iSeries uses the OS/400 journaling support to help its backup and restore strategy. You must be familiar with standard OS/400 backup and recovery methods, and with the use of journals and their associated journal receivers on OS/400, before reading this section. For information on these topics, see *OS/400 Backup and Recovery*.

To understand the backup and recovery strategy, you first need to understand how WebSphere MQ for iSeries organizes its data in the OS/400 file system and the integrated file system (IFS)

WebSphere MQ for iSeries holds its data in an individual library for each queue manager, and in stream files in the IFS file system.

The queue manager specific libraries contain journals, journal receivers, and objects required to control the work management of the queue manager. The IFS directories and files contain WebSphere MQ configuration files, the descriptions of WebSphere MQ objects, and the data they contain.

Every change to these objects, that is recoverable across a system failure, is recorded in a journal *before* it is applied to the appropriate object. This has the effect that such changes can be recovered by replaying the information recorded in the journal.

WebSphere MQ for iSeries journals

WebSphere MQ for iSeries uses journals in its operation to control updates to local objects. Each queue manager library contains a journal for that queue manager, which has the name `QMGRLIB/AMQAJRN`, where `QMGRLIB` is the name of the queue manager library.

`QMGRLIB` takes the name `QM`, followed by the name of the queue manager in a unique form. For example, a queue manager named `TEST` has a journal receiver library named `QMTEST`.

These journals have associated journal receivers that contain the information being journaled. These receivers are objects to which information can only be appended and will fill up eventually.

They also use up valuable disk space with out-of-date information. However, you can place the information in permanent storage to minimize this problem. One journal receiver is attached to the journal at any particular time. If the journal receiver reaches its predetermined threshold size, it is detached and replaced by a new journal receiver.

The journal receivers associated with the local WebSphere MQ for iSeries journal exist in each queue manager library, and adopt a naming convention as follows:

```
AMQArnnnnn
```

where

nnnnn
is decimal 00000 to 99999

r is decimal 0 to 9

The sequence of the journals is based on date. However, the naming of the next journal is based on the following rules:

1. AMQArnnnnn goes to AMQAr(nnnnn+1), and nnnnn wraps when it reaches 99999. For example, AMQA000000 goes to AMQA000001, and AMQA999999 goes to AMQA000000.
2. If a journal with a name generated by rule 1 already exists, the message CPI70E3 is sent to the QSYSOPR message queue and automatic receiver switching stops.

   The currently-attached receiver continues to be used until you investigate the problem and manually attach a new receiver.
3. If no new name is available in the sequence (that is, all possible journal names are on the system) you need to do both of the following:
   a. Delete journals no longer needed (see "Journal management" on page 84).
   b. Record the journal changes into the latest journal receiver using (RCDMQMIMG) and then repeat the previous step. This allows the old journal receiver names to be reused.

The AMQAJRN journal uses the MNGRCV(*SYSTEM) option to enable the operating system to automatically change journal receivers when the threshold is reached. For more information on how the system manages receivers, see *OS/400 Backup and Recovery*.

The journal receiver's default threshold value is 100 000 KB. This is set when the queue manager is created and is determined by the **MaxReceiverSize** value defined in the LogDefaults stanza of the mqs.ini file. See Chapter 9, "Configuring WebSphere MQ" on page 109 for further details on configuring the system.

If you need to change the size of journal receivers after the queue manager has been created, create a new journal receiver and set its owner to QMQM using the following commands:

```
CRTJRNRCV JRNRCV(QMGRLIB/AMQAnnnnnn) THRESHOLD(xxxxxx) +
   TEXT('MQM LOCAL JOURNAL RECEIVER')
CHGOBJOWN OBJ(QMGRLIB/AMQAnnnnnn) OBJTYPE(*JRNRCV) NEWOWN(QMQM)
```

where

QmgrLib
Is the name of your queue manager library

nnnnnnn
Is the next journal receiver in the naming sequence described

xxxxxx Is the new receiver threshold (in KB)

Note: The maximum size of the receiver is governed by the operating system. To check this value look at the THRESHOLD keyword on the CRTJRNRCV command.

Now attach the new receiver to the AMQAJRN journal with the command:

```
CHGJRN JRN(QMGRLIB/AMQAJRN) JRNRCV(QMGRLIB/AMQAnnnnnn)
```

See “Journal management” on page 84 for details on how to manage these journal receivers.

WebSphere MQ for iSeries journal usage

Persistent updates to message queues happen in two stages. The records representing the update are first written to the journal, then the queue file is updated.

The journal receivers can therefore become more up-to-date than the queue files. To ensure that restart processing begins from a consistent point, WebSphere MQ uses checkpoints.

A checkpoint is a point in time when the record described in the journal is the same as the record in the queue. The checkpoint itself consists of the series of journal records needed to restart the queue manager. For example, the state of all transactions (that is, units of work) active at the time of the checkpoint.

Checkpoints are generated automatically by WebSphere MQ. They are taken when the queue manager starts and shuts down, and after every 10 000 operations logged.

You can force a queue manager to take a checkpoint by issuing the RCDMQMIMG command against all objects on a queue manager and displaying the results, as follows:

```
RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES)
```

As the queues handle further messages, the checkpoint record becomes inconsistent with the current state of the queues.

When WebSphere MQ is restarted, it locates the latest checkpoint record in the log. This information is held in the checkpoint file that is updated at the end of every checkpoint. The checkpoint record represents the most recent point of consistency between the log and the data. The data from this checkpoint is used to rebuild the queues as they existed at the checkpoint time. When the queues are recreated, the log is then played forward to bring the queues back to the state they were in before system failure or close down.

To understand how WebSphere MQ uses the journal, consider the case of a local queue called `TESTQ` in the queue manager `TEST`. This is represented by the IFS file:

```
/QIBM/UserData/mqm/qmgrs/TEST/queues
```

If a specified message is put on this queue, and then retrieved from the queue, the actions that take place are shown in Figure 10.

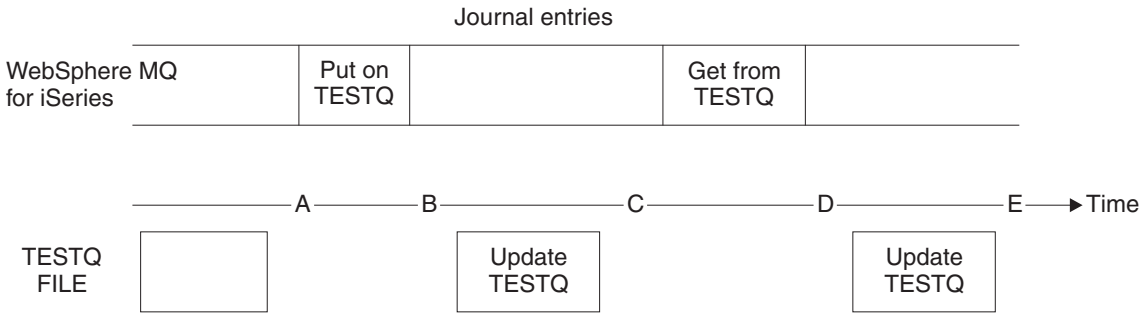

Figure 10. Sequence of events when updating MQM objects

The five points, A through E, shown in the diagram represent points in time that define the following states:

A The IFS file representation of the queue is consistent with the information contained in the journal.

B A journal entry is written to the journal defining a `Put` operation on the queue.

C The appropriate update is made to the queue.

D A journal entry is written to the journal defining a `Get` operation from the queue.

E The appropriate update is made to the queue.

The key to the recovery capabilities of WebSphere MQ for iSeries is that the user can save the IFS file representation of TESTQ as at time `A`, and subsequently recover the IFS file representation of TESTQ as at time `E`, simply by restoring the saved object and replaying the entries in the journal from time `A` onwards.

This strategy is used by WebSphere MQ for iSeries to recover persistent messages after system failure. WebSphere MQ remembers a particular entry in the journal receivers, and ensures that on startup it replays the entries in the journals from this point onwards. This startup entry is periodically recalculated so that WebSphere MQ only has to perform the minimum necessary replay on the next startup.

WebSphere MQ provides individual recovery of objects. All persistent information relating to an object is recorded in the local WebSphere MQ for iSeries journals. Any WebSphere MQ object that becomes damaged or corrupt can be completely rebuilt from the information held in the journal.

For more information on how the system manages receivers, see *OS/400 Backup and Recovery*.

Media images

A WebSphere MQ object of long duration can represent a large number of journal entries, going back to the point at which it was created. To avoid this overhead, WebSphere MQ for iSeries has the concept of a *media image* of an object.

This media image is a complete copy of the WebSphere MQ object recorded in the journal. If an image of an object is taken, the object can be rebuilt by replaying journal entries from this image onwards. The entry in the journal that represents the replay point for each WebSphere MQ object is referred to as its *media recovery entry*.

Images of the *CTLG object and the *MQM object are regularly taken, because these objects are required for WebSphere MQ to run at all.

Images of other objects are taken when convenient, particularly when the WebSphere MQ queue manager is ended. WebSphere MQ keeps track of the:
- Media recovery entry for each MQM object
- Oldest entry from within this set

WebSphere MQ automatically records an image of an object, if it finds a convenient point at which an object can be compactly described by a small entry in the journal. However, this might never happen for some objects, for example, queues that consistently contain large numbers of messages.

Rather than allow the date of the oldest media recovery entry to continue for an unnecessarily long period, use the WebSphere MQ command `RCDMQMIMG`, which enables you to take an image of selected objects manually.

Recovery from media images

WebSphere MQ automatically recovers some objects from their media image if it is found that they are corrupt or damaged. In particular, this applies to the special *MQM and *CTLG objects as part of the normal queue manager startup. If any syncpoint transaction was incomplete at the time of the last shutdown of the queue manager, any queue affected is also recovered automatically, in order to complete the startup operation.

You must recover other objects manually, using the WebSphere MQ command `RCRMQMOBJ`. This command replays the entries in the journal to re-create the WebSphere MQ object. Should a WebSphere MQ object become damaged, the only valid actions are to delete it or re-create it by this method. Note, however, that nonpersistent messages cannot be recovered in this fashion.

Backups of WebSphere MQ for iSeries data

For each queue manager, there are two types of WebSphere MQ backup to consider:

- Data and journal backup.

  To ensure that both sets of data are consistent, do this only after shutting down the queue manager.
- Journal backup.

  You can do this while the queue manager is active.

For both methods, you need to find the names of the queue manager IFS directory and the queue manager library. You can find these in the WebSphere MQ configuration file (mqs.ini). For more information, see "The QueueManager stanza" on page 113.

Use the procedures below to do both types of backup:

Data and journal backup of a particular queue manager

Note: Do not use a save-while-active request when the queue manager is running. Such a request cannot complete unless all commitment definitions with pending changes are committed or rolled back. If

this command is used when the queue manager is active, the channel connections might not end normally. Always use the procedure described below.

1. Create an empty journal receiver, using the command:
   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```
2. Use the RCDMQMIMG command to record an MQM image for all WebSphere MQ objects, and then force a checkpoint using the command:
   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```
3. End channels and ensure that the queue manager is not running. If your queue manager is running, stop it with the ENDMQM command.
4. Backup the queue manager library by issuing the following command:
   ```
   SAVLIB LIB(QMTEST)
   ```
5. Back up the queue manager IFS directories by issuing the following command:
   ```
   SAV DEV(...)  OBJ(('/QIBM/UserData/mqm/qmgrs/test'))
   ```

Journal backup of a particular queue manager

Because all relevant information is held in the journals, as long as you perform a full save at some time, partial backups can be performed by saving the journal receivers. These record all changes since the time of the full backup and are performed by issuing the following commands:

1. Create an empty journal receiver, using the command:
   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```
2. Use the RCDMQMIMG command to record an MQM image for all WebSphere MQ objects, and then force a checkpoint using the command:
   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```
3. Save the journal receivers using the command:
   ```
   SAVOBJ OBJ(AMQ*) LIB(QMTEST) OBJTYPE(*JRNRCV) .........
   ```

A simple backup strategy is to perform a full backup of the WebSphere MQ libraries every week, and perform a daily journal backup. This, of course, depends on how you have set up your backup strategy for your enterprise.

Journal management

As part of your backup strategy, take care of your journal receivers. It is useful to remove journal receivers from the WebSphere MQ libraries, in order to:

1. Release space; this applies to all journal receivers
2. Improve the performance when starting (STRMQM)
3. Improve the performance of recreating objects (RCRMQMOBJ)

Before deleting a journal receiver, be sure that:
1. You have a backup copy.
2. You no longer need the journal receiver.

Journal receivers can be removed from the queue manager library *after* they have been detached from the journals and saved, provided that they are available for restoration if needed for a recovery operation.

The concept of journal management is shown in Figure 11 on page 85.

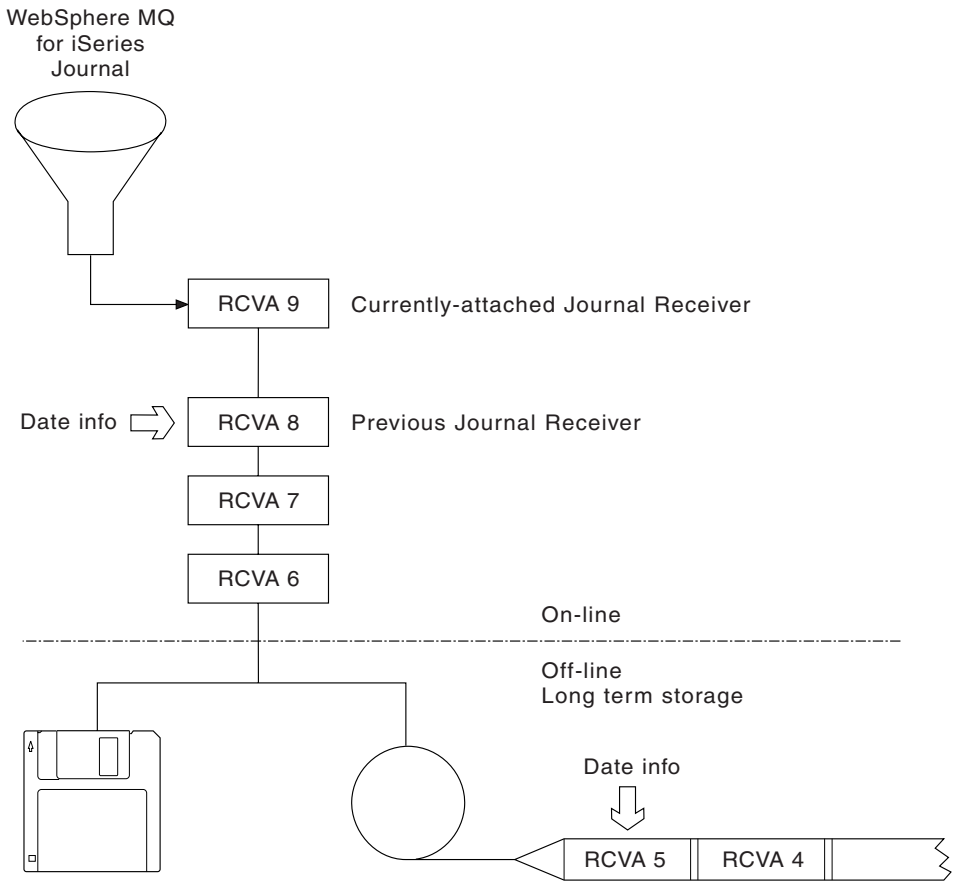

Figure 11. WebSphere MQ for iSeries journaling

It is important to know how far back in the journals WebSphere MQ is likely to need to go, in order to determine when a journal receiver that has been backed up can be removed from the queue manager library, and when the backup itself can be discarded.

To help determine this time, WebSphere MQ issues two messages to the queue manager message queue (QMQMMSG in the queue manager library) when:

- It starts up
- It changes a local journal receiver
- You use RCDMQIMG to force a checkpoint

These messages are:

AMQ7460
: Startup recovery point. This message defines the date and time of the startup entry from which WebSphere MQ replays the journal in the event of a startup recovery pass. If the journal receiver that contains this record is available in the WebSphere MQ libraries, this message also contains the name of the journal receiver containing the record.

AMQ7462
: Oldest media recovery entry. This message defines the date and time of the oldest entry to use to re-create an object from its media image.

  The journal receiver identified is the oldest one required. Any other WebSphere MQ journal receivers with older creation dates are no longer

needed. If only stars are displayed, you need to restore backups from the date indicated to determine which is the oldest journal receiver.

When these messages are logged, WebSphere MQ also writes a user space object to the queue manager library that contains only one entry: the name of the oldest journal receiver that needs to be kept on the system. This user space is called AMQJRNINF, and the data is written in the format:

```
JJJJJJJJJJLLLLLLLLLLYYYYMMDDHHMMSSmmm
```

where:

JJJJJJJJJJ
Is the oldest receiver name that WebSphere MQ still needs.

LLLLLLLLLL
Is the journal receiver library name.

YYYY Is the year of the oldest journal entry that WebSphere MQ needs.

MM Is the month of the oldest journal entry that WebSphere MQ needs.

DD Is the day of the oldest journal entry that WebSphere MQ needs.

HH Is the hour of the oldest journal entry that WebSphere MQ needs.

SS Is the seconds of the oldest journal entry that WebSphere MQ needs.

mmm Is the milliseconds of the oldest journal entry that WebSphere MQ needs.

When the oldest journal receiver has been deleted from the system, this user space contains asterisks (*) for the journal receiver name.

Note: Periodically performing `RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES)` can save startup time for WebSphere MQ and reduce the number of local journal receivers you need to save and restore for recovery.

WebSphere MQ for iSeries does not refer to the journal receivers unless it is performing a recovery pass either for startup, or for recreating an object. If it finds that a journal it requires is not present, it issues message `AMQ7432` to the queue manager message queue (QMQMMSG), reporting the time and date of the journal entry it requires to complete the recovery pass.

If this happens, restore all journal receivers that were detached after this date from the backup, in order to allow the recovery pass to succeed.

Keep the journal receiver that contains the startup entry, and any subsequent journal receivers, available in the queue manager library.

Keep the journal receiver containing the oldest `Media Recovery Entry`, and any subsequent journal receivers, available at all times, and either present in the queue manager library or backed-up.

When you force a checkpoint:

- If the journal receiver named in AMQ7460 is not advanced, this indicates that there is an incomplete unit of work that needs to be committed or rolled back.
- If the journal receiver named in AMQ7462 is not advanced, this indicates that there are one or more damaged objects.

Restoring a complete queue manager (data and journals)

If you need to recover one or more WebSphere MQ queue managers from a backup, perform the following steps.

1. Quiesce the WebSphere MQ queue managers.
2. Locate your latest backup set, consisting of your most recent full backup and subsequently backed up journal receivers.
3. Perform a RSTLIB operation, from the full backup, to restore the WebSphere MQ data libraries to their state at the time of the full backup, by issuing the following commands:

   ```
   RSTLIB LIB(QMQRLIB1) .........
   RSTLIB LIB(QMQRLIB2) .........
   ```

   If a journal receiver was partially saved in one journal backup, and fully saved in a subsequent backup, restore only the fully saved one. Restore journals individually, in chronological order.
4. Perform an RST operation to restore the WebSphere MQ IFS directories to the IFS file system, using the following command:

   ```
   RST DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/testqm'))  ...
   ```
5. Start the message queue manager. This replays all journal records written since the full backup and restores all the WebSphere MQ objects to the consistent state at the time of the journal backup.

If you want to restore a complete queue manager on a different machine, use the procedure below to restore everything from the queue manager library. (We use TEST as the sample queue manager name.)

1. `CRTMQM TEST`
2. `DLTLIB LIB(QMTEST)`
3. `RSTLIB SAVLIB(QMTEST) DEV(*SAVF) SAVF(QMGRLIBSAV)`
4. Delete the following IFS files:

   ```
   '/qibm/userdata/mqm/qmgrs/TEST/QMQMCHKPT'
   '/qibm/userdata/mqm/qmgrs/TEST/qmanager/QMQMOBJCAT'
   '/qibm/userdata/mqm/qmgrs/TEST/queues/SYSTEM.AUTH.DATA.QUEUE'
   ```
5. `STRMQM TEST`
6. `RCRMQMOBJ OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(TEST)`

Restoring journal receivers for a particular queue manager

The most common action is to restore a backed-up journal receiver to a queue manager library, if a receiver that has been removed is needed again for a subsequent recovery function.

This is a simple task, and requires the journal receivers to be restored using the standard OS/400 RSTOBJ command:

```
RSTOBJ OBJ(QMQMDATA/AMQA000005) OBJTYPE(*JRNRCV) .........
```

A series of journal receivers might need to be restored, rather than a single receiver. For example, `AMQA000007` is the oldest receiver in the WebSphere MQ libraries, and both `AMQA000005` and `AMQA000006` need to be restored.

In this case, restore the receivers individually in reverse chronological order. This is not always necessary, but is good practice. In severe situations, you might need to use the OS/400 command `WRKJRNA` to associate the restored journal receivers with the journal.

Backups of data

When restoring journals, the system automatically creates an attached journal receiver with a new name in the journal receiver sequence. However, the new name generated might be the same as a journal receiver you need to restore. Manual intervention is needed to overcome this problem; to create a new name journal receiver in sequence, and new journal before restoring the journal receiver.

For instance, consider the problem with saved journal AMQAJRN and the following journal receivers:

AMQA000000
AMQA100000
AMQA200000
AMQA300000
AMQA400000
AMQA500000
AMQA600000
AMQA700000
AMQA800000
AMQA900000

When restoring journal AMQAJRN to a queue manager library, the system automatically creates journal receiver AMQA000000. This automatically generated receiver conflicts with one of the existing journal receivers (AMQA000000) you want to restore, which you cannot restore.

The solution is:

1. Manually create the next journal receiver (see "WebSphere MQ for iSeries journals" on page 79):
   ```
   CRTJRNRCV JRNRCV(QMQRLIB/AMQA900001) THRESHOLD(XXXXX)
   ```
2. Manually create the journal with the above journal receiver:
   ```
   CRTJRN JRN(QMGRLIB/AMQAJRN) MNGRCV(*SYSTEM)  +
   JRNRCV(QMGRLIB/AMQA9000001) MSGQ(QMGRLIB/AMQAJRNMSG)
   ```
3. Restore the local journal receivers AMQA000000 to AMQA900000.

Performance considerations

If you use a large number of persistent messages or large messages in your applications, there is an associated overhead of journaling these messages.

This increases your system disk input/output. If this disk input/output becomes excessive, performance suffers.

Ensure that you have sufficient disk activation to cope with this possibility, or consider a separate ASP in which to hold your queue manager journal receivers. For more information, see *OS/400 V4R4M0 Backup and Recovery*.

Using SAVLIB to save WebSphere MQ libraries

You cannot use `SAVLIB LIB(*ALLUSR)` to save the WebSphere MQ libraries, because these libraries have names beginning with Q.

You can use `SAVLIB LIB(QM*)` to save all the queue manager libraries, but only if you are using a save device other than *SAVF. For `DEV(*SAVF)`, you must use a SAVLIB command for each and every queue manager library on your system.

Chapter 8. Analyzing problems

This chapter suggests reasons for problems you might have with WebSphere MQ for iSeries, to help you in problem determination. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

Do not confuse problem determination with problem solving although 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.

However, you might not always be able to solve a problem after determining its cause. For example:

- A performance problem might be caused by a limitation of your hardware.
- The cause of the problem might be in the WebSphere MQ for iSeries code. If this happens, you need to contact your IBM support center for a solution.

This chapter is divided into the following sections:

Preliminary checks

Before you start problem determination in detail, it is worth considering the facts to see if there is something obvious with 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 any of the following:

- Hardware
- Operating system
- Related software, for example, a language compiler
- The network
- The WebSphere MQ product
- Your WebSphere MQ application
- Other applications
- Site operating procedures

The sections that follow raise some fundamental questions that you need to consider.

As you go 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.

Preliminary checks

The following steps are intended to help you isolate the problem and are taken from the viewpoint of an WebSphere MQ application. Check all the suggestions at each stage.

1. Has WebSphere MQ for iSeries run successfully before?

   Yes Proceed to Step 2.

   No It is likely that you have not installed or set up WebSphere MQ correctly.

2. Has the WebSphere MQ application run successfully before?

   Yes Proceed to Step 3.

   No Consider the following:

   a. The application might have failed to compile or link, and fails if you attempt to invoke it. Check the output from the compiler or linker.

      Refer to the appropriate programming language reference manual, or the *WebSphere MQ Application Programming Guide*, for information on how to build your application.

   b. Consider the logic of the application. For example, do the symptoms of the problem indicate that a function is failing and, therefore, that a piece of code is in error.

      Check the following common programming errors:

      - Assuming that queues can be shared, when they are in fact exclusive.
      - Trying to access queues and data without the correct security authorization.
      - Passing incorrect parameters in an MQI call; if the wrong number of parameters is passed, no attempt can be made to complete the completion code and reason code fields, and the task is ended abnormally.
      - Failing to check return codes from MQI requests.
      - Using incorrect addresses.
      - Passing variables with incorrect lengths specified.
      - Passing parameters in the wrong order.
      - Failing to initialize *MsgId* and *CorrelId* correctly.

3. Has the WebSphere MQ application changed since the last successful run?

   Yes It is likely that the error lies in the new or modified part of the application. Check all the changes and see if you can find an obvious reason for the problem.

   a. Have all the functions of the application been fully exercised before?

      Could it be that the problem occurred when part of the application that had never been invoked before was used for the first time? If so, it is likely that the error lies in that part of the application. Try to find out what the application was doing when it failed, and check the source code in that part of the program for errors.

   b. If the program has run successfully before, check the current queue status and files that were being processed when the error occurred. It is possible that they contain some unusual data value that causes a rarely used path in the program to be invoked.

   c. The application received an unexpected MQI return code. For example:

- Does your application assume that the queues it accesses are shareable? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
- Have any queue definition or security profiles been changed? An MQOPEN call could fail because of a security violation; can your application recover from the resulting return code?

Refer to the information in the *WebSphere MQ Application Programming Reference* for your programming language for a description of each return code.

d. If you have applied any PTF to WebSphere MQ for iSeries, check that you received no error messages when you installed the PTF.

No Ensure that you have eliminated all the preceding suggestions and proceed to Step 4.

4. Has the server system remained unchanged since the last successful run?

Yes Proceed to "Problem characteristics".

No Consider all aspects of the system and review the appropriate documentation on how the change might have impacted the WebSphere MQ application. For example :
- Interfaces with other applications
- Installation of new operating system or hardware
- Application of PTFs
- Changes in operating procedures

Problem characteristics

Perhaps the preliminary checks have enabled you to find the cause of the problem. If so, you can probably now 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. Use the following questions as pointers to the problem. Answering the appropriate question, or questions, should lead you to the cause of the problem.

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which you do so:

- Is it caused by a command?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the `SYSTEM.ADMIN.COMMAND.QUEUE` has not been changed.
- Is it caused by a program? If so, does it fail in batch? Does it fail on all WebSphere MQ for iSeries systems, or only on some?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.
- Does the problem occur with any queue manager, or when connected to one specific queue manager?
- Does the problem occur with the same type of object on any queue manager, or only one particular object? What happens after this object has been cleared or redefined?

- Is the problem independent of any message persistence settings?
- Does the problem occur only when syncpoints are used?
- Does the problem occur only when one or more queue-manager events are enabled?

Is the problem intermittent?

An intermittent problem could be caused by failing to take into account the fact that processes can run independently of each other. For example, a program might issue an MQGET call, without specifying a wait option, before an earlier process has completed. You might also encounter this if your application tries to get a message from a queue while the call that put the message is in-doubt (that is, before it has been committed or backed out).

Problems with commands

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.

Queue managers and their associated object names are case sensitive. By default, OS/400 uses uppercase characters, unless you surround the name in quotes.

For example, `MYQUEUE` and `myqueue` translate to `MYQUEUE`, whereas `'myqueue'` translates to `myqueue`.

Does the problem affect all users of the WebSphere MQ for iSeries application?

If the problem affects only some users, look for differences in how the users configure their systems and queue manager settings.

Check the library lists and user profiles. Can the problem be circumvented by having `*ALLOBJ` authority?

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check the following:

- Is the connection between the two systems available, and has the intercommunication component of WebSphere MQ for iSeries started?

  Check that messages are reaching the transmission queue, the local queue definition of the transmission queue, and any remote queues.
- Have you made any network-related changes that might account for the problem or changed any WebSphere MQ for iSeries definitions?
- Can you distinguish between a channel definition problem and a channel message problem?

  For example, redefine the channel to use an empty transmission queue. If the channel starts correctly, the definition is correctly configured.

Does the problem occur only on WebSphere MQ

If the problem occurs only on this version of WebSphere MQ, check the appropriate database on RETAIN®, or the Web site `http://www.ibm.com/software/ts/mqseries/support/summary/400.html`, to ensure that you have applied all the relevant PTFs.

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, and so these are the times when load-dependent problems are most likely to occur. (If your WebSphere MQ for iSeries network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Have you failed to receive a response from a command?

If you have issued a command but you have not received a response, consider the following questions:

- Is the command server running?

  Work with the **DSPMQMCSVR** command to check the status of the command server.

  – If the response to this command indicates that the command server is not running, use the **STRMQMCSVR** command to start it.

  – If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.

- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See 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 100 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 attempting to receive reply messages.

- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?

- Are you using the *CorrelId* and *MsgId* fields correctly?

Problem characteristics

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 try and 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.

If you have still not identified the cause of the problem, see "Determining problems with WebSphere MQ applications".

Determining problems with WebSphere MQ applications

This section discusses problems you might encounter with WebSphere MQ applications, commands, and messages.

Are some of your queues working?

If you suspect that the problem occurs with only a subset of queues, select the name of a local queue that you think is having problems.

1. Display the information about this queue, using WRKMQMQSTS or DSPMQMQ.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, the queue is not being processed. Check that all applications are running normally.
   - If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
     - If triggering is being used:
       - Is the trigger monitor running?
       - Is the trigger depth too big?
       - Is the process name correct?
     - 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 (for example, because the applications need to be started, a connection has been disrupted, or because the MQOPEN call has failed for some reason).

If you cannot solve the problem, contact your IBM support center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues, check the following:

1. Check that the programs that should be putting messages to the remote queues have run successfully.

2. If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
3. If necessary, start the channel manually. See *WebSphere MQ Intercommunication* for information about how to do this.
4. Check the channel with a PING command.

See *WebSphere MQ Intercommunication* for information about how to define channels.

Does the problem affect messages?

This section deals with message problems, including:

- "Messages do not appear on the queue"
- "Messages contain unexpected or corrupted information" on page 96
- "Unexpected messages are received when using distributed queues" on page 96

Messages do not appear on the queue

If messages do not appear when you are expecting them, check for the following:

- Have you selected the correct queue manager, that is, the default queue manager or a named queue manager?
- Has the message been put on the queue successfully?
  - Has the queue been defined correctly, for example is MAXMSGLEN sufficiently large?
  - Can applications put messages on the queue (is the queue enabled for putting)?
  - Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Can you get the message 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 timeout interval long enough?
  - 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 if you can get other messages from the queue.
  - Can other applications get messages from the queue?
  - Was the message you are expecting defined as persistent?

    If not, and WebSphere MQ for iSeries has been restarted, the message will have been lost.

If you cannot find anything wrong with the queue, and the queue manager itself is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application start?

  If it should have been triggered, check that the correct trigger options were specified.
- Is a trigger monitor running?

- Was the trigger process defined correctly?
- Did it complete correctly?

  Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they might occasionally conflict with one another. For example, one transaction might issue an MQGET call with a buffer length of zero to find out the length of the message, and then issue a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction might have issued a successful MQGET call for that message, so the first application receives a completion code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multi-server environment must be designed to cope with this situation.

Consider that the message 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 contain unexpected or corrupted information".

Messages contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

- Has your application, or the application that put the message on to the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, a copyfile formatting the message might have been changed, in which case, re-compile both applications to pick up the changes. If one application has not been recompiled, the data appears 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 has used an alias queue, check that the alias points to the correct queue.
- Has the trigger information been specified correctly for this queue?

  Check that your application should have been started, or should a different application have been started?
- Has the CCSID been set correctly, or is the message format incorrect because of data conversion.

If these checks do not enable you to solve the problem, check your application logic, both for the program sending the message, and for the program receiving it.

Unexpected messages are received when using distributed queues

If your application uses distributed queues, consider the following points:

- Has distributed queuing been correctly installed on both the sending and receiving systems?
- Are the links available between the two systems?

Check that both systems are available, and connected to WebSphere MQ for iSeries. Check that the connection between the two systems is active.

- Is triggering set on in the sending system?
- Is the message you are waiting for a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

  This could mean that an application was unable to put the required message on to the queue. If this is so, check if the message has been put onto the undelivered-message queue.

  The dead-letter queue message header (dead-letter header structure) contains a reason or feedback code explaining why the message could not be put on to the target queue. See the *WebSphere MQ Application Programming Reference* or the *WebSphere MQ for iSeries Application Programming Reference (ILE RPG)*, as appropriate, for information about the dead-letter header structure.
- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap stops the distributed queuing component. See *WebSphere MQ Intercommunication* for more information about distributed queuing.

Obtaining diagnostic information

You can find diagnostic information about WebSphere MQ in the following places:

User's job log
The job log records the commands processed by the job and the messages returned from running those commands.

Reviewing the job log of a user who experiences a problem, by issuing the DSPJOBLOG command, identifies the WebSphere MQ commands issued and the sequence of those commands.

WebSphere MQ job log
WebSphere MQ specific jobs, for example, the command server and channel programs, run under the WebSphere MQ profile QMQM. If you have a problem in these areas, review these job logs by issuing the command `WRKSPLF QMQM` to display them.

System history log
Reviewing the history log, by issuing the DSPLOG command, displays information about the operation of the system and system status. This can be useful for identifying channel connection problems.

OS/400 message queue
It is useful to view messages sent to various OS/400 message queues using the DSPMSG command. Use the command `DSPMSG QSYSOPR` to check the system operator message queue, used for WebSphere MQ journaling messages, and job completion messages in particular.

Work with problems
Use the WRKPRB command to display descriptions of system problems. WebSphere MQ reports problems related to unusual usage and internal code by using this command.

Error logs in the IFS
See "Error logs" on page 100 for further information on using the error logs generated.

Generation of FFSTs
See "First-failure support technology (FFST)" on page 104 for further information on First Failure Support Technology™ and an example of a WebSphere MQ for iSeries FFST™ report.

Using WebSphere MQ for iSeries trace

Although you need to use certain traces on occasion, running the trace facility slows your systems.

You also need to consider to what destination you want your trace information sent.

Notes:

1. To run the WebSphere MQ for iSeries trace commands, you must have the appropriate authority.
2. Trace data is written to IFS only when trace is ended, with option *OFF.
3. Trace data remains in the system until you delete it from the IFS.

Trace usage

There are two stages in using trace:

1. Decide whether you want early tracing. Early tracing lets you trace the creation and startup of queue managers. Note, however, that early trace can easily generate large amounts of trace, because it is implemented by tracing **all** jobs for **all** queue managers. To enable early tracing, use TRCMQM with the TRCEARLY parameter set to *YES.
2. Start tracing WebSphere MQ work using `TRCMQM` *ON*. To stop the trace, you have two options:
   - `TRCMQM` *OFF*, to stop collecting trace records for a queue manager. The trace records are written to files in the `/QIBM/UserData/mqm/trace` directory.
   - `TRCMQM` *END*, to stop collecting trace records for **all** queue managers and to disable early trace. This option ignores the value of the TRCEARLY parameter.

You can also specify the level of detail you want, using the TRCLEVEL parameter set to:

*DFT For minimum-detail level for flow processing trace points.

*DETAIL
For high-detail level for flow processing trace points.

*PARMS
For default-detail level for flow processing trace points.

Selective trace

You can reduce the amount of trace data being saved, improving run-time performance, using the command `TRCMQM` with `F4=prompt`, then `F9` to customize the TRCTYPE and EXCLUDE parameters:

TRCTYPE
Specifies the type of trace data to store in the trace file. If you omit this parameter, all trace points except those specified in EXCLUDE are enabled.

EXCLUDE
Specifies the type of trace data to **omit** from the trace file. If you omit this parameter, all trace points specified in TRCTYPE are enabled.

The options available on both TRCTYPE and EXCLUDE are:

***ALL (TRCTYPE only)**
All the trace data as specified by the following keywords is stored in the trace file.

trace-type-list
You can specify more than one option from the following keywords, but each option can appear only once.

***API** Output data for trace points associated with the MQI and major queue manager components.

***CMTRY**
Output data for trace points associated with comments in the 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.

***CSFLOW**
Output data for trace points associated with processing flow in common services.

***LQMDATA**
Output data for trace points associated with internal data buffers in the local queue manager.

***LQMFLOW**
Output data for trace points associated with processing flow in the local queue manager.

***OTHDATA**
Output data for trace points associated with internal data buffers in other components.

***OTHFLOW**
Output data for trace points associated with processing flow in other components.

***RMTDATA**
Output data for trace points associated with internal data buffers in the communications component.

***RMTFLOW**
Output data for trace points associated with processing flow in the communications component.

***SVCDATA**
Output data for trace points associated with internal data buffers in the service component.

Diagnostic information

***SVCFLOW**
Output data for trace points associated with processing flow in the service component.

***VSNDATA**
Output data for trace points associated with the version of WebSphere MQ running.

Wrapping trace

Use the MAXSTG parameter to wrap trace, and to specify the maximum size of storage to be used for the collected trace records. The options are:

DFT* Trace wrapping is **not enabled. For each job, trace data is written to a file with the suffix .TRC until tracing is stopped.

maximum-K-bytes
Trace wrapping is enabled. When the trace file reaches its maximum size, it is renamed with the suffix .TRS, and a new trace file with suffix .TRC is opened. Any existing .TRS file is deleted. Specify a value in the range 1 through 16 000.

Formatting trace output

To format any trace output:

- Enter the QShell
- Enter the command

```
dspmqtrc [-t Format] [-h] [-s] [-o OutputFileName] InputFileName
```

where:

InputFileName
Is a **required** parameter specifying the name of the file containing the unformatted trace. For example `/QIBM/UserData/mqm/trace/AMQ12345.TRC`.

-t *FormatTemplate*
Specifies the name of the template file containing details of how to display the trace. The default value is `/QIBM/ProdData/mqm/lib/amqtrc.fmt`.

-h Omit header information from the report.

-s Extract trace header and put to stdout.

-o *output_filename*
The name of the file into which to write formatted data.

You can also specify `dspmqtrc *` to format all trace.

Error logs

WebSphere MQ uses a number of error logs to capture messages concerning the operation of WebSphere MQ itself, any queue managers that you start, and error data coming from the channels that are in use.

The location of the error logs depends on whether the queue manager name is known.

In the IFS:

- If the queue manager name is known and the queue manager is available, error logs are located in:

```
/QIBM/UserData/mqm/qmname/errors
```

- If the queue manager is not available, error logs are located in:
  /QIBM/UserData/mqm/&SYSTEM/errors

You can use the system utility EDTF to browse the errors directories and files. For example:

```
EDTF '/QIBM/UsedData/mqm/errors'
```

Alternatively, you can use option 23 against the queue manager from the WRKMQM panel.

Log files

At installation time, an &SYSTEM errors subdirectory is created in the IFS. The errors subdirectory can contain up to three error log files named:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files have the same names as the &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, that is /QIBM/UserData/mqm/*qmname*/errors.

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 name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the &SYSTEM errors subdirectory.

To examine the contents of any error log file, use your system editor, EDTF, to view the stream files in the IFS.

Notes:

1. Do ***not*** change ownership of these error logs.
2. If any error log file is deleted, it is automatically re-created when the next error message is logged.

Early errors

There are a number of special cases where the error logs have not yet been established and an error occurs. 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.

If both the WebSphere MQ configuration file and the DefaultPrefix attribute of the AllQueueManagers stanza are readable, errors are logged in the errors subdirectory of the directory identified by the DefaultPrefix attribute.

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national language enabled, with message catalogs installed in standard locations.

These messages are written to the joblog, 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.

An example WebSphere MQ error log

Figure 12 on page 103 shows a typical extract from a WebSphere MQ error log.

```
************Beginning of data**************
07/19/02  11:15:56 AMQ9411: Repository manager ended normally.

EXPLANATION:
Cause . . . . . :   The repository manager ended normally.
Recovery  . . . :   None.
Technical Description . . . . . :   None.
----------------------------------------------------------------------------
07/19/02  11:15:57 AMQ9542: Queue manager is ending.

EXPLANATION:
Cause . . . . . :   The program will end because the queue manager is quiescing.
Recovery  . . . :   None.
Technical Description . . . . . :   None.
----- amqrimna.c : 773 ------------------------------------------------------

07/19/02  11:16:00 AMQ8004: WebSphere MQ queue manager 'mick' ended.
EXPLANATION:
Cause . . . . . :   WebSphere MQ queue manager 'mick' ended.
Recovery  . . . :   None.
Technical Description . . . . . :   None.
----------------------------------------------------------------------------
07/19/02  11:16:48 AMQ7163: WebSphere MQ job number 18429 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18429 the CCSID is 37. The job name is
                    582775/MQUSER/AMQZXMA0.
Recovery  . . . :   None
----------------------------------------------------------------------------
07/19/02  11:16:49 AMQ7163: WebSphere MQ job number 18430 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18430 the CCSID is 0. The job name is
                    582776/MQUSER/AMQZFUMA.
Recovery  . . . :   None
----------------------------------------------------------------------------
07/19/02  11:16:49 AMQ7163: WebSphere MQ job number 18431 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18431 the CCSID is 37. The job name is
                    582777/MQUSER/AMQZXMAX.
Recovery  . . . :   None
----------------------------------------------------------------------------
07/19/02  11:16:50 AMQ7163: WebSphere MQ job number 18432 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18432 the CCSID is 37. The job name is
                    582778/MQUSER/AMQALMPX.
Recovery  . . . :   None
----------------------------------------------------------------------------
```

Figure 12. Extract from a WebSphere MQ 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 an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the provided browse sample application (amqsbcg) to browse messages on the queue using the MQGET call. The sample application steps through all the

messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue.

Problems might occur if you do not associate a dead-letter queue with each queue manager. For more information about dead-letter queues, see Chapter 6, "The WebSphere MQ dead-letter queue handler" on page 69.

First-failure support technology (FFST)

This section describes the role of first-failure support technology (FFST).

For OS/400, FFST information is recorded in a stream file in the
| `/QIBM/UserData/mqm/errors` directory, and, for non-threaded jobs, in the problem database accessed using the OS/400 command **WRKPRB**.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or a WebSphere MQ internal error.

The stream files are named `AMQ`*`nnnnn`*`.`*`mm`*`.FDC`, where:

nnnnn	Is the ID of the process reporting the error
mm	Is a sequence number, normally 0

| Some typical FFST data is shown in Figure 13 on page 105.
|

```
------------------------------------------------------------------------------
| WebSphere MQ First Failure Symptom Report                                  |
| =================================                                          |
|                                                                            |
| Date/Time         :- Friday June 21 18:40:34  2002                         |
| Host Name         :- WINAS12B.HURSLEY.IBM.COM                              |
| PIDS              :- 5733A38                                               |
| LVLS              :- 520                                                   |
| Product Long Name :- WebSphere MQ for iSeries                              |
| Vendor            :- IBM                                                   |
| Probe Id          :- XY353001                                              |
| Application Name  :- MQM                                                   |
| Component         :- xehAS400ConditionHandler                              |
| Build Date        :- May 10 2002                                           |
| UserID            :- 00000331 (MAYFCT)                                     |
| Program Name      :- STRMQM_R  MAYFCT                                      |
| Job Name          :- 020100/MAYFCT/STRMQM_R                                |
| Activation Group  :- 101 (QMQM) (QMQM/STRMQM_R)                            |
| Process           :- 00001689                                              |
| Thread            :- 00000001                                              |
| QueueManager      :- TEST.AS400.OE.P                                       |
| Major Errorcode   :- STOP                                                  |
| Minor Errorcode   :- OK                                                    |
| Probe Type        :- HALT6109                                              |
| Probe Severity    :- 1                                                     |
| Probe Description :- 0                                                     |
| Arith1            :- 1 1                                                   |
| Comment1          :-    00d0                                               |
------------------------------------------------------------------------------

MQM Function Stack
lpiSPIMQConnect
zstMQConnect
ziiMQCONN
ziiClearUpAgent
xcsTerminate
xlsThreadInitialization
xcsConnectSharedMem
xstConnSetInSPbyHandle
xstConnSharedMemSet
xcsFFST

MQM Trace History
              <-- xcsCheckProcess rc=xecP_E_INVALID_PID
              --> xcsCheckProcess
              <-- xcsCheckProcess rc=xecP_E_INVALID_PID
              --> xlsThreadInitialization
               --> xcsConnectSharedMem
                --> xcsRequestThreadMutexSem
                <-- xcsRequestThreadMutexSem rc=OK
                --> xihGetConnSPDetailsFromList
                <-- xihGetConnSPDetailsFromList rc=OK
                --> xstCreateConnExtentList
                <-- xstCreateConnExtentList rc=OK
                --> xstConnSetInSPbyHandle
                 --> xstSerialiseSPList
                  --> xllSpinLockRequest
                  <-- xllSpinLockRequest rc=OK
                 <-- xstSerialiseSPList rc=OK
                 --> xstGetSetDetailsFromSPByHandle
                 <-- xstGetSetDetailsFromSPByHandle rc=OK
                 --> xstConnSharedMemSet
                  --> xstConnectExtent
                   --> xstAddConnExtentToList
                   <-- xstAddConnExtentToList rc=OK
                  <-- xstConnectExtent rc=OK
                  --> xcsBuildDumpPtr
                   --> xcsGetMem
                   <-- xcsGetMem rc=OK
                  <-- xcsBuildDumpPtr rc=OK
                  --> xcsBuildDumpPtr
                  <-- xcsBuildDumpPtr rc=OK
                  --> xcsBuildDumpPtr
                  <-- xcsBuildDumpPtr rc=OK
                  --> xcsFFST

Process Control Block
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :8bba0:0:6d   E7C9C8D7  000004E0  00000699  00000000    XIHP...\...r....
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :8bbb0:1:6d   00000000  00000002  00000000  00000000    ................
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :8bbc0:2:6d   80000000  00000000  EC161F7C  FC002DB0    ...........0...¢
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :8bbd0:3:6d   80000000  00000000  EC161F7C  FC002DB0    ...........0...¢
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :8bbe0:4:6d   00000000  00000000  00000000  00000000    ................

Thread Control Block
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :1db0:20:6d   E7C9C8E3  00001320  00000000  00000000    XIHT............
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :1dc0:21:6d   00000001  00000000  00000000  00000000    ................
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :1dd0:22:6d   80000000  00000000  DD13C17B  81001000    .........A#a...
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :1de0:23:6d   00000000  00000046  00000002  00000001    ................
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :1df0:24:6d   00000000  00000000  00000000  00000000    ................


RecoveryIndex
SPP:0000 :1aefSTRMQM_R  MAYFCT    020100 :2064:128:6d   00000000                                   ....
```

Figure 13. FFST report

Notes:

1. The `MQM Trace History` section is a log of the 200 most recent function trace statements, and is recorded in the FFST report regardless of any TRCMQM settings.
2. The queue manager details are recorded only for jobs that are connected to a queue manager subpool.
3. When the failing component is `xehAS400ConditionHandler`, additional data is logged in the errors directory giving extracts from the joblog relating to the exception condition.

The function stack and trace history are used by IBM to assist in problem determination. In most cases there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center.

Performance considerations

This section discusses:

- General design considerations; see "Application design considerations"
- Specific performance problems; see "Specific performance problems" on page 107

Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making WebSphere MQ for iSeries calls are discussed in the following sections.

For more information about application design, see the *WebSphere MQ Application Programming Guide*.

Effect of message length

Although WebSphere MQ for iSeries allows messages to hold up to 100 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, 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 journaled. Journaling 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. The use of the MQGET call in this way affects the performance of your application.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the *BufferLength* field

set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the *MaxMsgLen* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be the maximum allowed by WebSphere MQ for iSeries, which could be greater than 2 GB.

Frequency of syncpoints

Programs that issue numerous MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently unusable, 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

An application might require a large number of threads. Each queue manager process is allocated a maximum allowable number of threads.

If some applications are troublesome, it could be due to their design using 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.

The maximum number of threads that OS/400 allows is 4095. However, the default is 64. WebSphere MQ makes available up to 63 threads to its processes.

Specific performance problems

This section discusses the problems of storage and poor performance.

Storage problems

If you receive the system message `CPF0907. Serious storage condition may exist` it is possible that you are filling up the space associated with the WebSphere MQ for iSeries queue managers.

Is your application or WebSphere MQ for iSeries running slowly?

If your application is running slowly, this could indicate that it is in a loop, or waiting for a resource that is not available.

This could also be caused by a performance problem. Perhaps it is because your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to you to occur at some other time.)

Performance considerations

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when certain queues are accessed.

QTOTJOB and QADLTOTJ are system values worth investigating.

The following symptoms might indicate that WebSphere MQ for iSeries is running slowly:

- If your system is slow to respond to MQSC commands.
- If repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.
- Is MQ trace running?

Chapter 9. Configuring WebSphere MQ

This chapter explains how to change the behavior of queue managers to suit your installation's needs.

You change WebSphere MQ configuration information by modifying the values specified on a set of configuration attributes (or parameters) that govern WebSphere MQ. You change these attributes by editing the **WebSphere MQ configuration files**.

This chapter:

- Describes the methods for configuring WebSphere MQ in "WebSphere MQ configuration files".
- Describes the attributes you can use to modify configuration information in "Attributes for changing WebSphere MQ configuration information" on page 111.
- Describes the attributes you can use to modify queue manager configuration information in "Changing queue manager configuration information" on page 113.
- Provides examples of mqs.ini and qm.ini files for WebSphere MQ for iSeries in "Example mqs.ini and qm.ini files" on page 120.

WebSphere MQ configuration files

You modify WebSphere MQ configuration attributes within:

- A WebSphere MQ configuration file (**mqs.ini**) to effect changes on the node as a whole. There is one mqs.ini file for each WebSphere MQ installation.
- 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.

Note that .ini files are stream files resident in the IFS.

A configuration file (which can be referred to as a ***stanza*** file) contains one or more stanzas, which are simply groups of lines in the .ini file that together have a common function or define part of a system, for example, log functions and channel functions.

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 the EDTF CL 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 possible.
- 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 a WebSphere MQ installation. It is created automatically during installation. In particular, the mqs.ini file is used to locate the data associated with each queue manager.

The mqs.ini file is stored in `/QIBM/UserData/mqm`

The mqs.ini file contains:
- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each queue manager
- Information identifying any API exits (see “API exits” on page 116 for more information)

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 `<mqmdata directory>/QMNAME/qm.ini`, where:

- <mqmdata directory> is `/QIBM/UserData/mqm` by default.
- `QMNAME` is the name of the queue manager to which the initialization file applies.

Notes:

1. You can change the `<mqmdata directory>` in the mqs.ini file.

2. 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**. See "Understanding WebSphere MQ queue manager library names" on page 165 for further information.

Attributes for changing WebSphere MQ configuration information

The following groups of attributes appear in mqs.ini:
- The AllQueueManagers stanza
- "The DefaultQueueManager stanza" on page 112
- "The ExitProperties stanza" on page 112
- "The QueueManager stanza" on page 113

There are also two stanzas associated with API exits, ApiExitCommon and ApiExitTemplate. For details on using these, see "Configuring API exits" on page 118.

Note: In the descriptions of the stanzas, the value underlined is the default value and the | symbol means *or*.

The AllQueueManagers stanza

The `AllQueueManagers` stanza can specify:
- The path to the qmgrs directory where the files associated with a queue manager are stored
- The path to the executable library
- The method for converting EBCDIC-format data to ASCII format

DefaultPrefix=*directory_name*

The path to the qmgrs directory, below which the queue manager data is kept.

If you change the default prefix for the queue manager, you must replicate the directory structure that was created at installation time.

In particular, you must create the qmgrs structure. Stop WebSphere MQ before changing the default prefix, and restart WebSphere MQ only after moving the structures to the new location and changing the default prefix.

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 the method WebSphere MQ is to use when converting 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 OS/400 for all EBCDIC to ASCII conversions.

Note that the effect of this type of conversion can vary from from language to language .

ISO
Specify ISO if you want:

- ISO CCSIDs to be converted using the TABLE method
- All other CCSIDs to be converted using the NL_TO_CF method.

Possible ISO CCSIDs are shown in Table 14.

Table 14. 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.

The DefaultQueueManager stanza

The `DefaultQueueManager` stanza specifies the default queue manager for the node.

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, you must alter the `DefaultQueueManager` attribute manually.

The ExitProperties stanza

The `ExitProperties` stanza specifies configuration options used by queue manager exit programs.

CLWLMode=SAFE|FAST
The cluster workload exit, CLWL, allows you to specify which cluster queue in the cluster is to be opened 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 to 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 bad return code.

The integrity of the queue manager is preserved.

Note: There is a possible performance overhead associated with running the CLWL exit in a separate process.

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. Run the CLWL exit in FAST mode only if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance overheads.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager fails and you run the risk of compromising the integrity of the queue manager.

The QueueManager stanza

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 queue manager library names" on page 165 for more information about name transformation.

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 `AllQueueManager` stanza in the mqs.ini file.

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.

Library=*name*
The name of the library where OS/400 objects pertinent to this queue manager, for example, journals and journal receivers, 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 library name.

Changing queue manager configuration information

The following groups of attributes can appear in a qm.ini file particular to a given queue manager, or used to override values set in mqs.ini.

- "The Log stanza" on page 114
- "The Channels stanza" on page 114
- "The TCP stanza" on page 116

There is also a stanza associated with API exits, ApiExitLocal. For details on using this, see "Configuring API exits" on page 118.

The Log stanza

The `Log` stanza specifies the log attributes for a particular queue manager. By default, these are inherited from the settings specified in the `LogDefaults` stanza in the mqs.ini file when the queue manager is created.

Only change attributes of this stanza if you want to configure a queue manager differently from others.

The values specified on the attributes in the qm.ini file are read when the queue manager is started. The file is created when the queue manager is created.

LogPath=*library_name*
The name of the library used to store journals and journal receivers for this queue manager.

LogReceiverSize
The journal receiver size.

The Channels stanza

The `Channels` stanza contains information about the 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. That is, there is no agent process.

STANDARD
Channels connect using STANDARD.

ThreadedListener=NO | YES
Whether to start RUNMQLSR (**YES**) or AMQCLMAA (**NO**) as a listener.

AdoptNewMCA=NO | SVR | SNDR | 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 ending 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 end the previous one.
2. If the previous channel server does not end 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.

You 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

SNDR
Adopt sender channels

RCVR
Adopt receiver channels

CLUSRCVR
Adopt cluster receiver channels

ALL
Adopt all channel types, except for 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 can behave in an unpredictable fashion with FASTPATH channels because of the internal design of the queue manager. 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 seconds, in the range 1 to 3600. The default value is 60.

AdoptNewMCACheck=QM|ADDRESS|NAME|ALL
The `AdoptNewMCACheck` attribute allows you to specify the type checking required when enabling the `AdoptNewMCA` attribute. It is important for you to perform all three of the following checks, if possible, 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, from the following:

QM
The listener process checks that the queue manager names match.

ADDRESS
The listener process checks the communications address, for example, the TCP/IP address.

NAME
The listener process checks that the channel names match.

ALL
The listener process checks 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.

The TCP stanza

This stanza specifies network protocol configuration parameters. They override the default attributes for channels.

Note: Only attributes representing changes to the default values need to be specified.

TCP
The following attributes can be specified:

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.

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
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 value for OS/400 is 255; the maximum is 512. If the backlog reaches the value of 512, the TCP/IP connection is rejected and the channel cannot start.

For MCA 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.

The `ListenerBacklog` attribute allows you to override the default number of outstanding requests for the TCP/IP listener.

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:

We give a brief introduction to writing API exits in "How to write an API exit". For detailed information about writing API exits, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

For detailed reference information about API exits, see the *WebSphere MQ System Administration 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 editing the configuration files, mqs.ini and qm.ini, and adding new stanzas that.

- 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 118. For a description of how API exits run, see "What happens when an API exit runs?" on page 118.

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. Within API exits, these calls take the general form:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

where `call` is the API call name without the MQ suffix (PUT, GET, and so on), `parameters` control the function of the exit, `context` describes the context in which the API exit was called, and `ApiCallParameters` represent the parameters to the MQI call,

What happens when an API exit runs?

The API exit routines to run are identified in stanzas in mqs.ini and qm.ini. 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.

You define your API exits in 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 120.

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.

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

Once you have created an exit as a service program, you must refer to it in the appropriate stanza in a configuration file using LIBRARY/PROGRAM syntax, as shown in the examples.

The mqs.ini file below contains the following stanzas:

ApiExitTemplate
This stanza defines an exit with the descriptive name `OurPayrollQueueAuditor`, module name `MYAUDIT`, 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 MYSECURE, and sequence number 1. The data passed is an instruction (CheckEverything).

mqs.ini

```
   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=MYLIB/MYAUDIT
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=MYLIB/MYSECURE
      Data=CheckEverything
```

The qm.ini file below contains an ApiExitLocal definition of an exit with the descriptive name ClientApplicationAPIchecker, module name MYCHECK, and sequence number 3.

qm.ini

```
   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=MYLIB/MYCHECK
      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.

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, stored in its own subdirectory.

The configuration files are stored in the IFS, as follows:

```
/QIBM/UserData/mqm/mqs.ini
/QIBM/UserData/mqm/qmgrs/<queue-manager-name>/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 using the EDTF CL command.

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.)

Example mqs.ini and qm.ini files

Figure 14 on page 121 shows an example of an mqs.ini file.

```
#********************************************************************#
#* Module Name: mqs.ini                                             *#
#* Type       : WebSphere MQ Configuration File                     *#
#* Function   : Define WebSphere MQ resources for the node          *#
#*                                                                  *#
#********************************************************************#
#* Notes      :                                                     *#
#* 1) This is an example WebSphere MQ configuration file            *#
#*                                                                  *#
#********************************************************************#
AllQueueManagers:
#********************************************************************#
#* The path to the qmgrs directory, below which queue manager data  *#
#* is stored                                                        *#
#********************************************************************#
DefaultPrefix=/QIBM/UserData/mqm

QueueManager:
   Name=saturn.queue.manager
   Prefix=/QIBM/UserData/mqm
   Library=QMSATURN.Q
   Directory=saturn!queue!manager

QueueManager:
   Name=pluto.queue.manager
   Prefix=/QIBM/UserData/mqm
   Library=QMPLUTO.QU
   Directory=pluto!queue!manager

DefaultQueueManager:
   Name=saturn.queue.manager
```

Figure 14. Example of a WebSphere MQ configuration file

Figure 15 shows how groups of attributes might be arranged in a queue manager configuration file.

```
#********************************************************************#
#* Module Name: qm.ini                                              *#
#* Type       : WebSphere MQ queue manager configuration file       *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                  *#
#********************************************************************#
#* Notes      :                                                     *#
#* 1) This file defines the configuration of the queue manager      *#
#*                                                                  *#
#********************************************************************#
Log:
   LogPath=QMSATURN.Q
   LogReceiverSize=65536

CHANNELS:
  MaxChannels = 20         ; Maximum number of channels allowed.
                           ; Default is 100.
  MaxActiveChannels = 10   ; Maximum number of channels allowed to be
                           ; active at any time.  The default is the
                           ; value of MaxChannels.

TCP:                       ; TCP/IP entries.
  KeepAlive = Yes          ; Switch KeepAlive on
```

Figure 15. Example queue manager configuration file

Notes:

1. WebSphere MQ on the node uses the default locations for queue managers and the journals.
2. The queue manager saturn.queue.manager is the default queue manager for the node. The directory for files associated with this queue manager has been automatically transformed into a valid file name for the file system.
3. 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 WebSphere MQ commands to fail. Also, applications cannot connect to a queue manager that is not defined in the WebSphere MQ configuration file.

Chapter 10. 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 "Installable services interface reference information" on page 130.

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 for iSeries, 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 for iSeries.
- 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 service available in WebSphere MQ for iSeries is the authorization 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 WebSphere MQ for iSeries. Table 15 summarizes support for the authorization service.

Table 15. Authorization service components summary

Supplied component	Function	Requirements
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 15. Authorization service components summary (continued)

Supplied component	Function	Requirements
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. 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.

Services and components are related as follows:

- 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.

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 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. 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 is automatically updated to include the appropriate stanzas for the authorization service and for the default component (the OAM).

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 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:

- Ensure that the header file `cmqzc.h` is included in your program.
- Create the shared library by compiling the program and linking it with the shared libraries `libmqm*` and `libmqmzf*`.

  Note: Because the agent can run in a threaded environment, you must build the OAM to run in a threaded environment. This includes using the threaded versions of `libmqm` and `libmqmzf`.
- 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.

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)"
- "Configuring authorization service stanzas" on page 128
- "Authorization service interface" on page 129

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 **WRKMQMAUT** (work with authority), **WRKMQMAUTD** (work with authority data), **DSPMQMAUT** (display object authority), **GRTMQMAUT** (grant object authority), **RVKMQMAUT** (revoke object authority), and **RFRMQMAUT** (refresh security).

The syntax of these commands and how to use them are described in the CL command help.

Authorization service

The OAM works with the *entity* of a principal or group.

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` define the authorization service to the queue manager. See "Configuring services and components" on page 126 for information about the types of stanza.

Configuring authorization service stanzas

On WebSphere MQ for iSeries:

Principal
Is an OS/400 system user profile.

Group Is an OS/400 system group profile.

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.

Each queue manager has its own queue manager configuration file. For example, the default path and file name of the queue manager configuration file for queue manager QMNAME is `/QIBM/UserData/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 `WRKENVVAR`. Any other *ServiceComponent* stanzas must be added manually.

For example, the following stanzas in the queue manager configuration file define two authorization service components:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Figure 16. WebSphere MQ for iSeries authorization service stanzas in qm.ini

The first 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 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.

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 127 for more information.

Installable services interface reference information

This section 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.

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_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 160 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_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_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 supported:

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_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 16. Fields in MQZAD

Field	Description	Page
StrucId	Structure identifier	160
Version	Structure version number	160
ProfileName	Profile name	160
ObjectType	Object type	161
Authority	Authority	161
EntityDataPtr	Address of MQZED structure identifying an entity	161
EntityType	Type of entity	161

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 17. Fields in MQZED

Field	Description	Page
StrucId	Structure identifier	163
Version	Structure version number	163
EntityNamePtr	Address of entity name	163
EntityDomainPtr	Address of entity domain name	163
SecurityId	Security identifier	163

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.

MQZED – Entity descriptor

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 */
};
```

Appendix A. WebSphere MQ names and default objects

This appendix describes the requirements for WebSphere MQ object names, queue manager name transformations, and lists the system default objects.

WebSphere MQ object names

The names of the followingWebSphere MQ objects can have up to 48 single-byte characters:

- Queue managers
- Queues
- Process definitions
- Namelists

The names of channels are restricted to 20 single-byte characters.

The characters that can be used for all WebSphere MQ names are:

- Uppercase A–Z
- Numerics 0–9
- Period (.)
- Underscore (_)
- Lowercase a–z (see note 1)
- Forward slash (/) (see note 1)
- Percent sign (%) (see note 1)

Notes:

1. Lowercase a–z, forward slash, and percent are special characters. If you use any of these characters in a name, the name must be enclosed in quotation marks. (Lowercase a–z characters are changed to uppercase if the name is not enclosed in quotation marks.)

   You cannot use lowercase characters on systems using EBCDIC Katakana.
2. Leading or embedded blanks are not allowed.

Understanding WebSphere MQ queue manager library names

A library is associated with each queue manager, and library names cannot be more than 10 characters long. However in WebSphere MQ, you can give a queue manager a name containing up to 48 characters.

For example, you can name a queue manager:

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

The queue manager name must therefore be transformed to give a unique library name. The rules for governing this transformation are:

1. Add QM to the start of the name
   - Truncate the name to 10 characters
   - Convert individual characters so that % becomes _, and / becomes #.

   After this transformation, `ACCOUNTING.SERVICES.QUEUE.MANAGER` becomes `QMACCOUNTI`.
2. If the name is still not valid, or the library exists:
   - Truncate the name transformed above to 8 characters

- Append a two-character numeric suffix

After this transformation, ACCOUNTING.SERVICES.QUEUE.MGR2 also becomes QMACCOUNTI, but, if a library already exists with this name, it becomes QMACCOUN00.

3. If the name is still not valid, increment the two-character numeric suffix by one and apply rule 2 on page 165 again.

   This suffix can be incremented up to 99 times to find a valid name.

Understanding WebSphere MQ IFS directories and files

The OS/400 Integrated File System (IFS) is used extensively by WebSphere MQ to store data. For more information about the IFS see the *Integrated File System Introduction*.

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 path to a queue manager directory is formed from the following:

- A prefix, which is defined in the queue manager configuration file, qm.ini. The default prefix is /QIBM/UserData/mqm.
- A literal, qmgrs.
- A coded queue manager name, which is the queue manager name transformed into a valid directory name. For example, the queue manager queue/manager is represented by queue&manager.

This process is referred to as name transformation.

IFS queue manager name transformation

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

For example, you can name a queue manager QUEUE/MANAGER/ACCOUNTING/SERVICES. In the same way that a library is created for each queue manager, each queue manager is also represented by a file. Because of variant codepoints in EBCDIC, there are limitations to the characters that can be used in the name. As a result, the names of IFS files representing objects are automatically transformed to meet the requirements of the file system.

Using the example of a queue manager with the name queue/manager, transforming the character / to &, and assuming the default prefix, the queue manager name in WebSphere MQ for iSeries becomes /QIBM/UserData/mqm/qmgrs/queue&manager.

Object name transformation

Object names are not necessarily valid file system names, so the object names might need to be transformed. The method used is different from that for queue manager names because, although there only a few queue manager names for each machine, there can be a large number of other objects for each queue manager. Only 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 DSPMQMOBJN command to view the transformed names for WebSphere MQ objects.

System and default objects

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

- The system objects are those WebSphere MQ objects required for the operation of 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 18 lists the system and default queue objects.
- Table 19 lists the system and default channel objects.
- Table 20 gives the system and default process object.
- Table 21 on page 168 gives the system and default namelist object.

Table 18. 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	Used by the object authority manager (OAM).
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.AUTHINFO.CRLLDAP	Default authentication information definition.
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.

Default objects

Table 18. System and default objects: queues (continued)

Object name	Description
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.

Table 19. System and default objects: channels

Object name	Description
SYSTEM.AUTO.RECEIVER	Dynamic receiver channel.
SYSTEM.AUTO.SVRCONN	Dynamic server-connection channel.
SYSTEM.DEF.CLNTCONN	Default client connection channel, used to supply default values for any attributes not specified when a CLNTCONN channel is created on a queue manager.
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.

Table 20. System and default objects: processes

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

Table 21. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist definition.

Appendix B. Sample resource definitions

This appendix contains the AMQSAMP4 sample OS/400 CL program.

```
/*********************************************************************/
 /*                                                                   */
 /* Program name: AMQSAMP4                                            */
 /*                                                                   */
 /* Description: Sample CL program defining MQM queues                */
 /*              to use with the sample programs                      */
 /*              Can be run, with changes as needed, after            */
 /*              starting the MQM                                     */
 /*                                                                   */
 /* Statement:     Licensed Materials - Property of IBM               */
 /*                                                                   */
 /*                5724-B41                                           */
 /*                (C) Copyright IBM Corporation 1993, 2002.          */
 /*                                                                   */
 /*********************************************************************/
 /*                                                                   */
 /* Function:                                                         */
 /*                                                                   */
 /*                                                                   */
 /*   AMQSAMP4 is a sample CL program to create or reset the          */
 /*   MQI resources to use with the sample programs.                  */
 /*                                                                   */
 /*   This program, or a similar one, can be run when the MQM         */
 /*   is started - it creates the objects if missing, or resets       */
 /*   their attributes to the prescribed values.                      */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*   Exceptions signaled:  none                                      */
 /*   Exceptions monitored: none                                      */
 /*                                                                   */
 /*   AMQSAMP4 has no parameters.                                     */
 /*                                                                   */
 /*********************************************************************/
            PGM


 /*********************************************************************/
 /*       EXAMPLES OF DIFFERENT QUEUE TYPES                           */
 /*                                                                   */
 /*   Create local, alias and remote queues                           */
 /*                                                                   */
 /*   Uses system defaults for most attributes                        */
 /*                                                                   */
 /*********************************************************************/
 /*   Create a local queue                                            */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.LOCAL')                +
                        QTYPE(*LCL)  REPLACE(*YES)                   +
                                                                     +
                        TEXT('Sample local queue') /* description */+
                        SHARE(*YES)                /* Shareable   */+
                        DFTMSGPST(*YES)  /* Persistent messages OK  */

 /*   Create an alias queue                                           */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.ALIAS')                +
                        QTYPE(*ALS)  REPLACE(*YES)                   +
```

```
                                                                      +
                        TEXT('Sample alias queue')                    +
                        DFTMSGPST(*YES) /* Persistent messages OK */+
                        TGTQNAME('SYSTEM.SAMPLE.LOCAL')

/*   Create a remote queue - in this case, an indirect reference    */
/*      is made to the sample local queue on OTHER queue manager    */
             CRTMQMQ    QNAME('SYSTEM.SAMPLE.REMOTE')               +
                        QTYPE(*RMT)  REPLACE(*YES)                   +
                                                                     +
                        TEXT('Sample remote queue')/* description */+
                        DFTMSGPST(*YES) /* Persistent messages OK */+
                        RMTQNAME('SYSTEM.SAMPLE.LOCAL')              +
                        RMTMQMNAME(OTHER)   /* Queue is on OTHER    */

/*   Create a transmission queue for messages to queues at OTHER    */
/*   By default, use remote node name                               */
             CRTMQMQ    QNAME('OTHER') /* transmission queue name */+
                        QTYPE(*LCL)  REPLACE(*YES)  +
                        TEXT('transmision queue to OTHER')  +
                        USAGE(*TMQ)    /* transmission queue      */

/********************************************************************/
/*       SPECIFIC QUEUES AND PROCESS USED BY SAMPLE PROGRAMS        */
/*                                                                  */
/*   Create local queues used by sample programs                    */
/*   Create MQI process associated with sample initiation queue     */
/*                                                                  */
/********************************************************************/
/*   General reply queue                                            */
             CRTMQMQ    QNAME('SYSTEM.SAMPLE.REPLY')                +
                        QTYPE(*LCL)  REPLACE(*YES)                   +
                                                                     +
                        TEXT('General reply queue')                  +
                        DFTMSGPST(*YES)  /* Persistent messages OK  */

/*   Queue used by AMQSINQA                                         */
             CRTMQMQ    QNAME('SYSTEM.SAMPLE.INQ')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                   +
                                                                     +
                        TEXT('queue for AMQSINQA')                   +
                        SHARE(*YES)               /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                      +
                        TRGENBL(*YES)  /* Trigger control on      */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')          +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*   Queue used by AMQSSETA                                         */
             CRTMQMQ    QNAME('SYSTEM.SAMPLE.SET')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                   +
                                                                     +
                        TEXT('queue for AMQSSETA')                   +
                        SHARE(*YES)               /* Shareable  */ +
                        DFTMSGPST(*YES)/* Persistent messages OK */ +
                                                                     +
                        TRGENBL(*YES)  /* Trigger control on     */ +
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.SETPROCESS')          +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*   Queue used by AMQSECHA                                         */
             CRTMQMQ    QNAME('SYSTEM.SAMPLE.ECHO')                 +
```

```
                        QTYPE(*LCL)  REPLACE(*YES)                +
                                                                  +
                        TEXT('queue for AMQSECHA')                +
                        SHARE(*YES)               /* Shareable  */ +
                        DFTMSGPST(*YES)/* Persistent messages OK */ +
                                                                  +
                        TRGENBL(*YES)  /* Trigger control on     */ +
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')      +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*   Initiation Queue used by AMQSTRG4, sample trigger process      */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.TRIGGER') +
                        QTYPE(*LCL)  REPLACE(*YES)   +
                        TEXT('trigger queue for sample programs')

/*   MQI Processes associated with triggered sample programs        */
/*                                                                  */
/*****    Note - there are versions of the triggered samples   ******/
/*****      in different languages - set APPID for these       ******/
/*****      process to the variation you want to trigger       ******/
/*                                                                  */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')       +
                        REPLACE(*YES)                             +
                                                                  +
                        TEXT('trigger process for AMQSINQA')      +
                        ENVDATA('JOBPTY(3)') /* Submit parameter */ +
         /** Select the triggered program here   **/               +
                        APPID('AMQSINQA')    /* C     */          +
              /*        APPID('AMQ0INQA')    /* COBOL */          +
              /*        APPID('AMQ3INQ4')    /* RPG - ILE */

            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.SETPROCESS')       +
                        REPLACE(*YES)                             +
                                                                  +
                        TEXT('trigger process for AMQSSETA')      +
                        ENVDATA('JOBPTY(3)') /* Submit parameter */ +
         /** Select the triggered program here   **/               +
                        APPID('AMQSSETA')    /* C     */          +
              /*        APPID('AMQ0SETA')    /* COBOL */          +
              /*        APPID('AMQ3SET4')    /* RPG - ILE */

            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')      +
                        REPLACE(*YES)                             +
                                                                  +
                        TEXT('trigger process for AMQSECHA')      +
                        ENVDATA('JOBPTY(3)') /* Submit parameter */ +
         /** Select the triggered program here   **/               +
                        APPID('AMQSECHA')    /* C     */          +
              /*        APPID('AMQ0ECHA')    /* COBOL */          +
              /*        APPID('AMQ3ECH4')    /* RPG - ILE */

/********************************************************************/
/*   Normal return.                                                 */
/********************************************************************/
            RETURN
            ENDPGM

/********************************************************************/
/* END OF AMQSAMP4                                                  */
/********************************************************************/
```

Appendix C. Quiescing WebSphere MQ and MQSeries systems

This appendix explains how to end gracefully (quiesce) a WebSphere MQ for iSeries or MQSeries for AS/400 system. There are two different processes:

1. "Quiescing MQSeries for AS/400 V5.1 systems"
2. "Quiescing MQSeries for AS/400 V5.2 and WebSphere MQ for iSeries systems"

Quiescing MQSeries for AS/400 V5.1 systems

To quiesce an MQSeries for AS/400 Version 5.1 system:

1. Sign on to a new interactive MQSeries for AS/400 session, ensuring that you are not accessing any MQSeries objects.
2. Ensure that you have:
   - *ALLOBJ authority , or object management authority for the QMQM library
   - Sufficient authority to use the ENDSBS command
   - *USE authority for the following programs:

     ```
     QMQM/AMQIQEM4
     QMQM/AMQSTOP4
     QMQM/AMQSPECA
     ```
3. Advise all users that you are going to stop MQSeries for AS/400.
4. Quiesce queue managers by running the AMQSTOP4 program, as follows:

   ```
   CALL QMQM/AMQSTOP4 PARM(*ALL *CNTRLD 15)
   ```

   If this fails, you can force a stop by issuing:

   ```
   CALL QMQM/AMQSTOP4 PARM(*ALL *FORCE 15)
   ```

Quiescing MQSeries for AS/400 V5.2 and WebSphere MQ for iSeries systems

To quiesce MQSeries for AS/400 V5.2 and WebSphere MQ for iSeries systems:

1. Sign on to a new interactive MQSeries for AS/400 or WebSphere MQ for iSeries session, ensuring that you are not accessing any objects.
2. Ensure that you have:
   - *ALLOBJ authority , or object management authority for the QMQM library
   - Sufficient authority to use the ENDSBS command
3. Advise all users that you are going to stop MQSeries for AS/400.
4. How you then proceed depends on whether you want to shut down (quiesce) a single queue manager (where others might exist) (see "Shutting down a single queue manager") or all the queue managers (see "Shutting down all queue managers" on page 175).

Shutting down a single queue manager

There are three types of shutdown:

- "Planned shutdown" on page 174
- "Unplanned shutdown" on page 174
- "Shutdown under abnormal conditions" on page 174

In the procedures that follow, we use a sample queue manager name of QMgr1 and a sample subsystem name of SUBX. Replace these with your own.

Planned shutdown

1. One hour before shutdown, execute:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```

2. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```

   If QMgr1 does not end, the channel or applications are probably busy.
3. If you need to shut down QMgr1 immediately, execute the following:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Unplanned shutdown

1. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   If QMgr1 does not end, the channel or applications are probably busy.
2. If you need to shut down QMgr1 immediately, execute the following:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Shutdown under abnormal conditions

1. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   If QMgr1 does not end, continue with step 3 providing that:
   - QMgr1 is in its own subsystem, or
   - You can end all queue managers that share the same subsystem as QMgr1. Use the unplanned shutdown procedure for all such queue managers.
2. When you have taken all the steps in the procedure for all the queue managers sharing the subsystem (SUBX in our examples), execute:

   ```
   ENDSBS SUBX *IMMED
   ```

   If this command fails to complete, shut down all queue managers, using the unplanned shutdown procedure, and IPL your machine.

   Warning

   Do not use ENDJOBABN for MQSeries or WebSphere MQ jobs that fail to end as result of ENDJOB or ENDSBS, unless you are prepared to IPL your machine immediately after.
3. Start the subsystem by executing:

   ```
   STRSBS SUBX
   ```

4. Shut the queue manager down immediately, by executing:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(10)
   ```

5. Restart the queue manager by executing:

   ```
   STRMQM MQMNAME(QMgr1)
   ```

   If this fails, and you:

- Have restarted your machine with an IPL, or
- Have only a single queue manager

Tidy up MQSeries or WebSphere MQ shared memory by executing:

```
ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
ENDCCTJOB(*YES) TIMEOUT(15)
```

before repeating step 5.

If you still have problems restarting your queue manager, contact IBM support. Any further action you might take could damage the queue manager, leaving MQSeries or WebSphere MQ unable to recover.

Shutting down all queue managers

There are three types of shutdown:
- "Planned shutdown"
- "Unplanned shutdown"
- "Shutdown under abnormal conditions" on page 176

The procedures are almost the same as for a single queue manager, but using `*ALL` instead of the queue manager name where possible, and otherwise using a command repeatedly using each queue manager name in turn. Throughout the procedures, we use a sample queue manager name of `QMgr1` and a sample subsystem name of `SUBX`. Replace these with your own.

Planned shutdown

1. One hour before shutdown, execute:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```

   Repeat this for every queue manager that you want to shut down.
2. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

   If any queue manager does not end within a reasonable time (for example 10 minutes), proceed to step 3.
3. To shut down all queue managers immediately, execute the following:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Unplanned shutdown

1. To shut down a queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

   If queue managers do not end, the channel or applications are probably busy.
2. If you need to shut down the queue managers immediately, execute the following:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Shutdown under abnormal conditions

1. To shut down the queue managers, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.
2. End the subsystems (SUBX in our examples), by executing:

   ```
   ENDSBS SUBX *IMMED
   ```

   Repeat this for every subsystem that you want to shut down; separate commands can run in parallel.

   If this command fails to complete, IPL your machine.

   Warning

   Do not use ENDJOBABN for MQSeries or WebSphere MQ jobs that fail to end as result of ENDJOB or ENDSBS, unless you are prepared to IPL your machine immediately after.
3. Start the subsystems by executing:

   ```
   STRSBS SUBX
   ```

   Repeat this for every subsystem that you want to start.
4. Shut the queue managers down immediately, by executing:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```
5. Restart the queue managers by executing:

   ```
   STRMQM MQMNAME(QMgr1)
   ```

   Repeat this for every queue manager that you want to start.

   If you still have problems restarting any queue manager, contact IBM support. Any further action you might take could damage the queue managers, leaving MQSeries or WebSphere MQ unable to recover.

Appendix D. 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.

Notices

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 are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	CICS	FFST
First Failure Support Technology	IBM	IBMLink
iSeries	MQSeries	OS/400
WebSphere	z/OS	

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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, and service names may be trademarks or service marks of others.

Index

D

E

F

G

H

P

Q

R

S

T

U

V

W

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.

SC34-6070-00