MQSeries® for AS/400®

System Administration

V5.1

SC34-5558-00

MQSeries® for AS/400®

System Administration

V5.1

SC34-5558-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 119.

First edition (March 2000)

This edition applies to the following product:

MQSeries for AS/400 Version 5 Release 1 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures vii

Tables ix

About this book xi
Who this book is for xi
What you need to know to understand this book . . xi
How to use this book xi

Chapter 1. Introduction to MQSeries . . 1
MQSeries and message queuing 1
Time-independent applications 1
Message-driven processing 1
Messages and queues 2
What is a message? 2
What is a queue? 2
Objects 3
Object names 4
Managing objects 4
Object attributes 4
MQSeries queue managers. 4
MQSeries queues 5
Process definitions 8
Channels 8
Clusters 8
Namelists 9
System default objects 9
Clients and servers 9
MQSeries applications in a client-server environment 10
Extending queue manager facilities 10
User exits 10
Security 10
Transactional support 10

Chapter 2. Managing MQSeries for AS/400 using CL commands 13
MQSeries applications 13
MQSeries for AS/400 CL commands 13
Starting a local queue manager 15
Creating MQSeries objects 15
Examples of creating a local queue 16
Examples of creating a remote queue 18
Creating a transmission queue 19
Creating an initiation queue 20
Creating an alias queue 20
Creating a model queue 20
Altering queue manager attributes. 20
Working with local queues 20
Defining a local queue. 20
Defining a dead-letter queue 21
Displaying default object attributes 22
Copying a local queue definition 22
Changing local queue attributes 22
Clearing a local queue 22
Deleting a local queue 23
Working with alias queues 23
Defining an alias queue 23
Using other commands with alias queues . . . 24
Working with model queues. 24
Defining a model queue 24
Using other commands with model queues. . . 25
Managing objects for triggering. 25
Defining an application queue for triggering . . 25
Defining an initiation queue 26
Creating a process definition 26
Displaying your process definition 26
Communicating between two systems 27

Chapter 3. Alternative methods for MQSeries administration 29
Local and remote administration 29
Performing administrative tasks using MQSC commands. 30
MQSC command files 30
Performing administrative tasks using PCF commands. 30
Attributes in MQSC and PCFs 31
Escape PCFs 31
Using the MQAI to simplify the use of PCFs . . 31
Using the MQSeries Explorer 32
What you can do with the MQSeries Explorer . . 33
Prerequisite software 33
Required definitions for administration 33
Managing the command server for remote administration 34
Starting the command server 34
Displaying the status of the command server . . 34
Stopping a command server 35

Chapter 4. Work management 37
Description of MQSeries Tasks 38
MQSeries work management objects 38
How MQSeries uses the work management objects 39
The MQSeries message queue 40
Configuring Work Management 41

Chapter 5. Protecting MQSeries objects 43
Security considerations 43
Understanding the Object Authority Manager . . . 44
Resources you can protect with the OAM . . . 44
MQSeries authorities 44
Granting MQSeries authorities to MQSeries objects 44
Understanding the authorization specification tables 47
MQI authorizations. 47
Administration authorizations 50
Authorizations for MQSC commands in escape PCFs 50

Authorizations for different types of object . . . 52
Object Authority Manager guidelines 53
Queue manager directories 53
Queues 53
Alternate-user authority 54
Context authority 54
Remote security considerations 54
Channel command security 55

Chapter 6. The MQSeries dead-letter queue handler 57
Invoking the DLQ handler 57
The DLQ handler rules table 58
Control data 58
Rules (patterns and actions) 59
Rules table conventions 62
Processing the rules table 63
Ensuring that all DLQ messages are processed . 64
An example DLQ handler rules table 65

Chapter 7. Instrumentation events . . . 67
What are instrumentation events? 67
Why use events? 68
Types of event 69
Event notification through event queues 70
Enabling and disabling events 70
Event messages 71

Chapter 8. Backup, recovery, and restart 73
MQSeries for AS/400 journals 73
MQSeries for AS/400 journal usage 75
Media images 76
Recovery from media images 77
Backups of MQSeries for AS/400 data 77
Journal management 78
Restoring a complete queue manager (data and journals) 80
Restoring journal receivers for a particular queue manager 80
Performance considerations 81

Chapter 9. Analyzing problems 83
Preliminary checks 83
Problem characteristics 85
Can the problem be reproduced? 85
Is the problem intermittent? 86
Problems with commands 86
Does the problem affect all users of the MQSeries for AS/400 application? 86
Does the problem affect specific parts of the network? 86
Does the problem occur only on MQSeries V5R1 87
Does the problem occur at specific times of the day? 87
Have you failed to receive a response from a command? 87
Determining problems with MQSeries applications 88
Are some of your queues working? 88
Does the problem affect only remote queues? . . 88
Does the problem affect messages? 89
Receiving unexpected messages when using distributed queues 90
Obtaining diagnostic information 91
Using MQSeries for AS/400 trace 92
Formatting trace output 93
Error logs 94
Log files 94
Early errors 95
Operator messages 95
An example MQSeries error log 95
Dead-letter queues 96
First-failure support technology (FFST) 97
Performance considerations 98
Application design considerations 98
Number of threads in use 100
Specific performance problems 100

Chapter 10. Configuring MQSeries 101
MQSeries configuration files 101
Editing configuration files 101
The MQSeries configuration file, mqs.ini . . . 102
Queue manager configuration files, qm.ini . . 102
Attributes for changing MQSeries configuration information 103
The AllQueueManagers stanza 103
The DefaultQueueManager stanza 104
The ExitProperties stanza 104
The QueueManager stanza 105
Changing queue manager configuration information 106
The Log stanza 106
The Channels stanza 106
The TCP stanza. 108
Example mqs.ini and qm.ini files 109

Appendix A. MQSeries names and default objects 111
MQSeries object names 111
Understanding MQSeries queue manager library names 111
Understanding MQSeries IFS directories and files 112
IFS Queue manager name transformation . . . 112
Object name transformation 112
System and default objects 113

Appendix B. Sample resource definitions 115

Appendix C. Notices 119
Trademarks 121

Glossary of terms and abbreviations 123

Bibliography 133
MQSeries cross-platform publications 133
MQSeries platform-specific publications 135
Softcopy books 136

BookManager format 136
HTML format 136
Portable Document Format (PDF) 136
PostScript format 136
Windows Help format 136
MQSeries information available on the Internet . . 136
Related publications 136

Index 137

Sending your comments to IBM . . . 141

Figures

1. Create MQM Queue initial panel 17
2. Work with MQM Queues panel 17
3. Extract from the MQSC command file, myprog.in 30
4. Display MQM Command Server panel 35
5. An example rule from a DLQ handler rules table 59
6. Understanding instrumentation events . . . 68
7. Monitoring queue managers across different platforms, on a single node 69
8. Sequence of events when updating MQM objects 75
9. MQSeries for AS/400 journaling 79
10. Extract from an MQSeries error log 96
11. FFST report 98
12. Example of an MQSeries configuration file 109
13. Example queue manager configuration file 110

Tables

1. MQSeries tasks. 38
2. Work management objects 38
3. Authorizations for MQI calls. 45
4. Authorizations for context calls 45
5. Authorizations for MQSC and PCF calls 45
6. Authorizations for generic operations 46
7. Security authorization needed for MQI calls 48
8. MQSC commands and security authorization needed 50
9. PCF commands and security authorization needed 51
10. Specifying authorizations for different object types. 52
11. List of possible ISO CCSIDs. 104
12. System and default objects - queues 113
13. System and default objects - channels 114
14. System and default objects - processes 114

About this book

This book applies to MQSeries for AS/400, V5.1.

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

This book describes the system administration aspects of MQSeries for AS/400, V5.1, 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 MQSeries is described in the *MQSeries for AS/400, V5.1 Quick Beginnings* book.

Post-installation configuration of a distributed queuing network is described in the *MQSeries Intercommunication* book.

Who this book is for

This book is intended for system administrators and system programmers who manage the configuration and administration tasks for MQSeries. It is also useful to application programmers who must have some understanding of MQSeries administration tasks.

What you need to know to understand this book

To use this book, you should have a good understanding of the IBM operating system for the AS/400, and of the utilities associated with it. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

For a summary of the new function introduced in MQSeries for AS/400, V5.1, see the *MQSeries for AS/400, V5.1 Quick Beginnings* book.

How to use this book

This book is divided into the following sections:

- The use of MQSeries for AS/400 using CL commands. This is the preferred method of operation.
- An overview of other methods of administering MQSeries for AS/400, V5.1.
- The various features of the product.
- A glossary and a bibliography at the back of the book.

Chapter 1. Introduction to MQSeries

This chapter introduces the MQSeries for AS/400 Version 5.1 product from an administrator's perspective, and describes the basic concepts of MQSeries and messaging. It contains these sections:

- "MQSeries and message queuing"
- "Messages and queues" on page 2
- "Objects" on page 3
- "System default objects" on page 9
- "Clients and servers" on page 9
- "Extending queue manager facilities" on page 10
- "Security" on page 10
- "Transactional support" on page 10

MQSeries and message queuing

MQSeries allows application programs to use **message queuing** to participate in message-driven processing. Application programs can communicate across different platforms by using the appropriate message queuing software products. For example, HP-UX and OS/390 applications can communicate through MQSeries for HP-UX and MQSeries for OS/390 respectively. The applications are shielded from the mechanics of the underlying communications.

MQSeries 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 *MQSeries Application Programming Reference* manual.

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 so that 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.

MQSeries 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 MQSeries. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference* manual.

Message lengths

The maximum length a message can be is 100 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length may be limited by:

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

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

What is a queue?

A **queue** is a data structure used to store messages. The messages may be 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 is responsible for maintaining the queues it owns and for storing all the messages it receives onto the appropriate queues.

The maximum size of a queue is 2 GB. For information about planning the amount of storage you require for queues, see the *MQSeries Planning Guide* or visit the following web site for platform-specific performance reports:

`http://www.software.ibm.com/ts/mqseries/txppacs/txpm1.html`

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

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

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

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

- **Predefined queues** are created by an administrator using the appropriate MQSeries commands. Predefined queues are permanent; they exist independently of the applications that use them and survive MQSeries 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 MQSeries 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 MQSeries **objects**.

In MQSeries Version 5.1, the object types include queue managers, queues, process definitions, channels, clusters, and namelists.

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 the *MQSeries Intercommunication* manual.
- Creating *clusters* of queue managers to simplify the overall administration process, or to achieve workload balancing. This is described in detail in the *MQSeries Queue Manager Clusters* manual.

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

- "Chapter 2. Managing MQSeries for AS/400 using CL commands" on page 13
- "Chapter 3. Alternative methods for MQSeries administration" on page 29

Object names

The naming convention adopted for MQSeries 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 should be 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 MQSeries, 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 "MQSeries object names" on page 111.

Managing objects

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

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

- MQSeries for AS/400 CL commands
- MQSeries commands (MQSC), 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
- MQSeries Administration Interface (MQAI) calls in a program

For more information about these methods, see "Chapter 3. Alternative methods for MQSeries administration" on page 29.

You can also administer MQSeries for AS/400 from a Windows NT machine using the MQSeries Explorer (see "Using the MQSeries Explorer" on page 32).

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 MQSeries, 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.

MQSeries 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 may exist on a remote machine across the network, or may exist on the same machine as the local queue manager.

MQSeries for AS/400, V5.1 supports multiple queue managers on the same machine.

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

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

MQSeries queues

Queues are defined to MQSeries using:

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

Note: The MQSeries 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 *MQSeries Command Reference* manual or the *MQSeries Programmable System Management* manual.

Using queue objects

There are four types of queue object available in MQSeries. 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 the *MQSeries Queue Manager Clusters* manual.
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 merely change the alias queue definition to reflect the name of the new queue to which the alias resolves.

   An alias queue is not a queue, but an object that you can use to access another queue.
4. **A model queue object** 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 may be temporary queues, which do not survive product restarts, or permanent queues, which do.

Specific local queue types and their uses

MQSeries 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 may 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 25 For more information about triggering, see the *MQSeries 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 the *MQSeries Intercommunication* book.
- **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 Version 5.1 of MQSeries for AIX, AS/400, HP-UX, OS/2, Warp, Sun Solaris, and Windows NT.

  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. In order 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 the *MQSeries Queue Manager Clusters* manual.
- **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 MQSeries commands for processing. These commands are then retrieved by an MQSeries 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** The MQSeries Version 5 products support 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 may inform an administrator or initiate some remedial action if the event indicates a problem. Note: Trigger events are quite different from instrumentation events in that trigger events are not caused by the same conditions, and do not generate event messages.

  For more information about instrumentation events, see the *MQSeries Programmable System Management* manual.

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries 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 MQSeries for AS/400 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 may exist on the same, or different, platforms. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another, complementary one, at the queue manager that is to receive them.

Use the MQSeries for AS/400 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 the *MQSeries Intercommunication* manual.

Clusters

In a traditional MQSeries 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, channels, and queue definitions.

For information about clusters, see the *MQSeries Queue Manager Clusters* book.

Namelists

A namelist is an MQSeries object that contains a list of other MQSeries 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 MQSeries object.

Use the MQSeries for AS/400 CRTMQMNL CL command, the MQSC command DEFINE NAMELIST, or the PCF command Create Namelist to create a namelist definition.

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 you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

Clients and servers

MQSeries supports client-server configurations for MQSeries applications.

An *MQSeries client* is a part of the MQSeries 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: MQSeries for AS/400, V5.1 cannot act as a client.

An *MQI server* is a queue manager that provides queuing services to one or more clients. All the MQSeries objects, for example queues, exist only on the queue

manager machine, that is, on the MQI server machine. A server can support normal local MQSeries applications as well.

For more information about creating channels for clients and servers, see the *MQSeries Intercommunication* book.

For information about client support in general, see the *MQSeries Clients* book.

MQSeries applications in a client-server environment

When linked to a server, client MQSeries 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 straightforward to set up
- It is straightforward to manage
- It has a low resource footprint

You must link your applications to the appropriate client libraries. See the *MQSeries Clients* book 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 mechanism for you to insert your own code into a queue manager function. The user exits supported include:

- **Channel exits** These exits change the way that channels operate. Channel exits are described in the *MQSeries Intercommunication* book
- **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 *MQSeries 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 the *MQSeries Queue Manager Clusters* book. The exit is supported in the following environments: AIX, AS/400, HP-UX, OS/2, Sun Solaris, Windows NT, and OS/390.

Security

In MQSeries for AS/400, V5.1 security is provided by the Object Authority Manager (OAM) component. See “Chapter 5. Protecting MQSeries objects” on page 43 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 succeeded while another failed then data integrity would be lost.

Transactional support

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 then 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 MQSeries queue manager. Here, syncpoint coordination is provided by the queue manager itself using a dual-phase commit process and use of the new MQI calls, MQBACK and MQCMIT.

MQSeries for AS/400 is **not** XA-compliant but is able to support and participate in global units of work coordinated by the AS/400 COMMIT and ROLLBACK commands.

About this book

Chapter 2. Managing MQSeries for AS/400 using CL commands

This chapter gives an overview of working with MQSeries for AS/400 from the AS/400 command line, together with some suggested operations.

MQSeries applications

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

- Recovery purposes
- Maintenance
- Rolling out MQSeries applications

You can do this by either:

- Creating CL programs to generate your MQSeries definitions for the AS/400, or
- Creating MQSC text files as SRC members to generate your MQSeries definitions using the cross-platform MQSeries command language.

MQSeries for AS/400 CL commands

The commands can be grouped as follows:

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

MQSeries applications

- Dead-Letter Queue Handler Command
  - STRMQMDLQ, Start MQSeries Dead-Letter Queue Handler
- Media Recovery Commands
  - RCDMQMIMG, Record MQM Object Image
  - RCRMQMOBJ, Recreate MQM Object
- MQSeries Command
  - STRMQMMQSC, Start MQSC Commands
- 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
- 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 managers
- Security Commands
  - DSPMQMAUT, Display MQM Object Authority
  - GRTMQMAUT, Grant MQM Object Authority
  - RVKMQMAUT, Revoke MQM Object Authority
- Trace Commands
  - TRCMQM, Trace MQM Job
- Transaction Commands
  - WRKMQMTRN, Work with MQSeries Transactions
  - RSVMQMTRN, Resolve MQSeries Transaction
- Trigger Monitor Command
  - STRMQMTRM, Start Trigger Monitor

Starting a local queue manager

You must:

1. Ensure that the MQSeries subsystem is running (using the command `STRSBS QMQM/QMQM`), and that the job queue associated with that subsystem is not held. By default, the MQSeries subsystem and job queue are both named QMQM in library QMQM.
2. Create a local queue manager by issuing the **CRTMQM** command from an AS/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 be only 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.
3. Start a local queue manager by issuing the **STRMQM** command from an AS/400 command line.

You can stop a queue manager by issuing the **ENDMQM** command from the AS/400 command line, and control a queue manager by issuing other MQSeries commands from an AS/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 MQSeries for AS/400) exist to enable such operations.

The local queue administrator cannot stop a remote queue manager.

Creating MQSeries objects

The following tasks suggest various ways in which you can use MQSeries for AS/400, from the command line.

There are two online methods to create MQSeries objects, which are:

1. Using a Create command:
   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:
   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, you can:
1. Use the Create MQM Queue (**CRTMQMQ**) command
2. Use the Work with MQM Queues (**WRKMQMQ**) command

Creating a local queue using the CRTMQMQ command

1. Type **CRTMQMQ** on the command line and press the `PF4` key.
2. On the Create MQM Queue panel, type the name of the queue 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 will be displayed, see Figure 1, with the fields containing the default values. You may 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 made any changes to the 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. 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.
3. Press the Enter key and Figure 2 is displayed.

```
                              Work with MQM Queues

Queue Manager Name . . .   MY.QUEUE.MANAGER_____________________       _____


Type options, press Enter.
  2=Change   3=Copy    4=Delete   5=Display   6=Clear   14=Display authority
  15=Grant authority   16=Revoke authority

Opt  Name                                              Type  Text
     TEST.QUEUE.LCL                                    *LCL  This is a text
     TKR.TEST.LCL                                      *LCL
     TKR.TEST.RMT                                      *RMT  REMOTE Q FOR
     TKR.TEST.XMIT.C2M                                 *LCL  XMITQ CORSAIR TO



                                                                      Bottom
Parameters for options 2, 3, 5, 14, 15, 16 or command
===> ____________________________________________________________________
F3=Exit   F4=Prompt      F5=Refresh   F6=Create   F9=Retrieve   F12=Cancel
F16=Repeat position to   F17=Position to          F20=Right     F21=Print
```

Figure 2. Work with MQM Queues panel

4. 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 16 for instructions on how to create the queue.

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

Examples of creating a remote queue

You 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 the *MQSeries Intercommunication* book.

This section describes how to define a remote queue for each of the three uses.

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, you:

1. Display the Create MQM Queue panel.
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 the *MQSeries Intercommunication* manual for further information.

To define a remote queue as a queue manager object, you:

1. Display the Create MQM Queue panel.
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 may 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, you define a remote queue with the same name as the reply-to queue.

See the *MQSeries Intercommunication* manual for further information.

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

1. Display the `Create MQM Queue` panel.
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.

Applications can put messages directly on a message queue, or they can be put there indirectly, for example, through a remote queue definition.

To create a transmission queue, you:

1. Display the `Create MQM Queue` panel.
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 *MQSeries Application Programming Guide*.

Creating an alias queue

You 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, you:

1. Display the `Create MQM Queue` panel.
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

You 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.

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 22.

Defining a dead-letter queue

Each queue manager should 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)

MQSeries 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 MQSeries dead-letter queue handler" on page 57.

Displaying default object attributes

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

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

Copying a local queue definition

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

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

This command creates a queue with the same attributes as our original queue orange.local.queue, rather than those of the system default local queue.

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

```
CPYMQMQ FROMQ('orange.local.queue') TOQ('third.queue') 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 are copying the queue attributes only. You are not copying 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 20, 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 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 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 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.

Working with alias queues

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

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

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

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

Defining an alias queue

The following command creates an alias queue:

```
CRTMQMQ QNAME('my.alias.queue') QTYPE(*ALS) TGTQNAME('yellow.queue')
```

This command redirects MQI calls that specify my.alias.queue to the queue yellow.queue. The command does not create the target queue; the MQI calls fail if the queue yellow.queue does not exist at run time.

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

```
CHGMQMQ QNAME('my.alias.queue') TGTQNAME('magenta.queue')
```

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:

Working with alias queues

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

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

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

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

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 queue aliases, in the same way that you use these attributes with local queues.

Using other commands with alias queues

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

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

DSPMQMQ 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 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.

Note: 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 QNAME('green.model.queue')

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

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

Managing objects for triggering

MQSeries 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 *MQSeries Application Programming Guide*.

This section describes how to set up the required objects to support triggering on MQSeries.

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 greater on the local queue motor.insurance.queue, as follows:

```
CRTMQMQ 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:

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

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

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

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

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

TRGENBL(*YES)
Is the trigger attribute value.

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

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

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

Defining an initiation queue

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

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

Where:

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

TEXT('Insurance request message processing')
Is 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)
Is the type of application to be started.

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

USRDATA('open, close, 235')
Is 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:

```
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 MQSeries for AS/400 systems, using CL commands, so that they can communicate with one another.

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

Carry out the following procedure:

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

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

2. Start this queue manager.

```
STRMQM     MQMNAME(QMGRA1)
```

3. Define the MQSeries 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 which points to a Queue called TARGETB              */
/* TARGETB belongs to Queue Manager QMGRB1 on SYSTEMB               */
 CRTMQMQ    QNAME(TARGETB.ON.QMGRB1) QTYPE(*SDR) +
              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 MQSeries 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.

   Note: This example uses the default port of 1414.

```
STRMQMLSR MQMNAME(QMGRB1)
```

Distributed queuing example

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 will be sent to queue TARGETB on SYSTEMB.

Check your messages by issuing the command:

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

Chapter 3. Alternative methods for MQSeries administration

You normally use the native AS/400 CL commands to perform administrative tasks. See “Chapter 2. Managing MQSeries for AS/400 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 the various methods, and includes the following topics:

- “Local and remote administration”
- “Performing administrative tasks using MQSC commands” on page 30
- “Performing administrative tasks using PCF commands” on page 30
- “Using the MQSeries Explorer” on page 32
- “Managing the command server for remote administration” on page 34

Local and remote administration

You administer MQSeries objects locally or remotely.

Local administration means carrying out administration tasks on any queue managers you have defined on your local system. In MQSeries, you can consider this as local administration because no MQSeries 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.

MQSeries supports administration from a single point through what is known as *remote administration*. All 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 “Performing administrative tasks using PCF commands” on page 30.
2. Writing a program using the MQAI, which actually sends out PCF messages. See “Using the MQAI to simplify the use of PCFs” on page 31.
3. Use the MQSeries Explorer, available with MQSeries for Windows NT, which allows you to use a graphical user interface (GUI) and generates the correct PCF messages. See “Using the MQSeries Explorer” on page 32.

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.

Performing administrative tasks using MQSC commands

You use MQSeries commands (MQSC) 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 AS/400 CL command. This is a batch method only, taking its input from a SRC PHYSICAL file in the AS/400 library system. The default name for this source physical file is QMQSC.

MQSC command files

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

Figure 3 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 3. Extract from the MQSC command file, myprog.in

For portability among MQSeries environments, you are recommended to 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 OS/390.

The *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

Performing administrative tasks using PCF commands

The purpose of MQSeries 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 the MQSC facility.

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

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

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

Briefly, these are some of the things the application programmer must specify to create a PCF command message:

Message descriptor

This is a standard MQSeries 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 the *MQSeries Programmable System Management* manual.

Attributes in MQSC and PCFs

Object attributes specified in MQSC 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, 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 the *MQSeries Programmable System Management* manual.

Using the MQAI to simplify the use of PCFs

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

Using PCFs

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.

The MQAI can be used:

- **To simplify the use of PCF messages** The MQAI is an easy way to administer MQSeries; 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.
- **To 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 will wait 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, see the *MQSeries Administration Interface Programming Guide and Reference* book.

For more information about PCFs in general, see the *MQSeries Programmable System Management* manual.

Using the MQSeries Explorer

The MQSeries Explorer is an application that runs under the Microsoft® Management Console (MMC) on Windows® NT version 4.0. It provides a graphical user interface for controlling MQSeries resources in an MQSeries network and is provided only with MQSeries for Windows NT V5.1.

The platforms and levels of MQSeries which can be administered using the MQSeries Explorer are described in “Prerequisite software” on page 33.

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 MQSeries Explorer from the First Steps application, or from the Windows NT Start prompt.

The configuration steps you must perform on remote MQSeries queue managers to allow the MQSeries Explorer to administer them are outlined in “Required definitions for administration” on page 33.

This section contains the following topics:

- “What you can do with the MQSeries Explorer” on page 33
- “Prerequisite software” on page 33
- “Required definitions for administration” on page 33

What you can do with the MQSeries Explorer

With the MQSeries Explorer, you can:

- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of MQSeries 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.
- 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.

Prerequisite software

Before you can use the MQSeries Explorer, you must have the following installed on your Windows NT computer:

- The Microsoft Management Console Version 1.1 or higher (installed as part of MQSeries for Windows NT 5.1 installation)
- Internet Explorer Version 4.01 (SP1) (installed as part of MQSeries for Windows NT 5.1 installation)

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

The MQSeries Explorer handles the differences in the capabilities between the different command levels and platforms. However, if it encounters a value which it does not recognize as an attribute for an object, you won’t be able to change the value of that attribute.

Required definitions for administration

Ensure that you have satisfied the following requirements before attempting to use the MQSeries Explorer to manage MQSeries on an AS/400 machine. Check that:

1. A command server is running for **any** queue manager being administered, starting on the AS/400 by the STRMQMCSVR CL command.
2. A suitable TCP/IP listener exists for every remote queue manager. This will be the MQSeries listener started by the STRMQMLSR command.
3. The server connection channel, called SYSTEM.ADMIN.SVRCONN, exists on every remote queue manager. This channel is created automatically when you issue a CRTMQM command.

   This channel is mandatory for every remote queue manager being administered. Without it, remote administration is not possible.

Required definitions

For further information on the MQSeries Explorer, see the *MQSeries System Administration* manual supplied with your MQSeries for Windows NT 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. This situation should be avoided, if at all possible.

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

Starting the command server

To start the command server use this CL command:

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

You must issue this command on the target machine. If the command server is running, the panel shown in Figure 4 on page 35 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 4. Display MQM Command Server panel

Stopping a command server

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

```
ENDMQMCSVR('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).

Command server remote administration

Chapter 4. Work management

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

Warning to users

You are strongly recommended **not** to alter MQSeries work management objects unless you fully understand the concepts of OS/400 and MQSeries work management.

Internal MQSeries jobs use threads – do **not** change any parameters in the objects described in this chapter.

Before reading this chapter you should familiarize yourself with the concepts of work management on the AS/400. You are recommended to look at the *OS/400 Work Management* manual, paying particular attention to the sections on "Job Starting and Routing" and "Batch Jobs".

During normal operations, an MQSeries queue manager starts a number of batch jobs to perform different tasks. By default these AS/400 batch jobs run in the QMQM subsystem that is created when MQSeries is installed.

Work Management refers to the process of tailoring MQSeries 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.

Work management is carried out by creating or changing the job descriptions associated with the MQSeries jobs, and is configurable for:

- An entire MQSeries installation
- Individual Queue managers
- Individual jobs for individual Queue Managers

Description of MQSeries 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 MQSeries subsystem. The jobs are described briefly in Table 1, to help you decide how to prioritize them.

Table 1. MQSeries tasks.

Job name	Function
AMQZXMA0	The execution controller is the first job started by the queue manager. It deals with MQCONN requests, and starts agent processes to process MQSeries API calls
AMQZLAA0	Queue manager agents perform the bulk of the work for applications that connect to the queue manager using MQCNO_STANDARD_BINDING
AMQALMPX	The checkpoint processor periodically takes journal checkpoints
AMQRRMFA	Repository manager for clusters
RUNMQCHL	This Sender Channel job will be started for each sender channel
RUNMQCHI	The Channel Initiator
AMQPCSEA	PCF command processor handles PCF and remote administration requests
AMQCRS6B	LU62 Receiver channel and client connection. (See note)
RUNMQLSR	TCP/IP Channel listener
Note: The LU62 receiver job runs in the communications subsystem and takes its run-time properties from the routing entry and communications entry that are used to start the job. See the *MQSeries Intercommunication* book for more details.	

MQSeries work management objects

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

Sample job descriptions are provided for two of the MQSeries batch jobs. If no specific job description is provided for an MQSeries job it runs with the default job description QMQMJOBD.

The work management objects that are supplied when you install MQSeries are listed in Table 2.

Table 2. Work management objects

Name	Type	Description
QMQM	*SBSD	The subsystem in which all MQSeries jobs run.
QMQM	*JOBQ	The job queue attached to the supplied subsystem
QMQMMSG	*MSGQ	The default message queue for MQSeries jobs.
QMQMRUN20	*CLS	A class description for high priority MQSeries jobs
QMQMRUN35	*CLS	A class description for medium priority MQSeries jobs
QMQMRUN50	*CLS	A class description for low priority MQSeries jobs
AMQZLAA0	*JOBD	The job description that is used by the MQSeries agent processes
AMQZXMA0	*JOBD	The job description that is used by MQSeries execution controllers
QMQMJOBD	*JOBD	The default MQSeries job description - used if there is not a specific job description for a job

How MQSeries uses the work management objects

To understand how you can configure work management, you should first understand how job descriptions are used by MQSeries.

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

- The job queue on which the job will be queued and, therefore, on which subsystem the job will run.
- The routing data used to start the job and, therefore, the class that the job uses for its run-time parameters.
- The output queue that the job will use for print files.

The process of starting an MQSeries job can be considered in three steps:

1. MQSeries selects a job description.

   MQSeries 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 MQSeries queue manager library names" on page 111 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 MQSeries 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 will 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 MQSeries jobs do not appear to be starting, make sure that the subsystem is running and the job queue is not held,

The MQSeries message queue

An MQSeries message queue, QMQMMSG, is created in each queue manager library. Operating system messages are sent to this queue when queue manager jobs end and MQSeries sends messages to the queue. For example, to report which journal receivers are needed at startup. It is a good idea to 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 MQSeries 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. You issue the **STRMQM** command for queue manager TESTQM.
2. MQSeries searches the queue manager library QMTESTQM, firstly for job description AMQZXMA0, and then job description QMQMJOBD.

   Neither of these job descriptions exist, so MQSeries looks for job description AMQZXMA0 in the product library QMQM. This job description does exist, so it is used to submit the job.
3. The job description uses the MQSeries 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 machine 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 will be 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. MQSeries searches the queue manager library QMTESTQM, firstly for job description AMQALPMX, and then job description QMQMJOBD.

   Neither of these job descriptions exist so MQSeries 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 will always be used for MQSeries jobs that do not have their own job description.
2. The job description uses the MQSeries 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 machine 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 will be 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 MQSeries job descriptions determine the run-time attributes of MQSeries jobs.

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

The key to the flexibility of MQSeries Work Management lies in the two—tier way that MQSeries searches for job descriptions:

- If you create or change job descriptions in a queue manager library, those changes will override the global job descriptions in QMQM but the changes will be *local* and affect that particular queue manager alone.
- If you create or change global job descriptions in the QMQM library, those job descriptions will 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 respectively, 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 MQSeries 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 will 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 will now:

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

2. 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)
   ```

   All jobs for queue manager TESTQM2 will now:

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

   To ensure that jobs are routed and prioritized correctly you can either:

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

  Note that this option works only if the maximum active jobs option for job queue QBATCH is set to *NOMAX.

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

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

   a. Create an output queue, for example:

      ```
      CRTOUTQ OUTQ(MYLIB/CHCKPTLOGS)
      ```

   b. Create a global duplicate of the QMQM/QMQMJOBD job description, using the name of the MQSeries 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 will be written to the job log.

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

   All MQSeries AMQALMPX jobs, for all queue managers, will 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 will now be written to output queue CHKPTLOGS in library MYLIB.

Chapter 5. Protecting MQSeries objects

Security for MQSeries for AS/400 changes significantly with Version 5 Release 1. Security is implemented using the MQSeries Object Authority Manager (OAM).

Security considerations

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

1. You should grant and revoke authorities to the MQSeries for AS/400 commands using the AS/400 **GRTOBJAUT** and **RVKOBJAUT** commands.
2. During installation of MQSeries for AS/400 the following special user profiles are created:

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

   QMQMADM
   Is intended to be used as a group profile for administrators of MQSeries. The group profile gives access to CL commands and MQSeries resources.

   NOBODY
   Is intended for internal product-only features.
3. If you are sending channel commands to remote queue managers, you must 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 55.
4. It is not essential for your user profile to belong to group QMQMADM to issue:
   - PCF commands, including Escape PCFs, from an administration program
   - MQI calls from an application program.
5. 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 the queue manager is restarted.
6. You should 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**)
   - Trace MQSeries (**TRCMQM**)
7. Channel definitions contain a security exit program specification. Channel creation and modification requires special considerations. Details of security, concerning exits, are given in the *MQSeries Intercommunication* book.

8. You need to be aware that 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 MQSeries 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 MQSeries 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 may be granted different kinds of access authority to the same object. For example, for a specific queue, one group may be allowed to perform both put and get operations; another group may be allowed only to browse the queue (MQGET with browse option). Similarly, some groups may have get and put authority to a queue, but are not allowed to alter or delete the queue.

MQSeries for AS/400 provides commands to grant, revoke, and display the authority that an application, or user, has to do the following:
- Issue MQSeries for AS/400 commands
- Perform operations on MQSeries for AS/400 objects

MQSeries authorities

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

Granting MQSeries authorities to MQSeries objects

You must either grant authority to MQSeries for AS/400 objects using the **GRTMQMAUT** command, or revoke the authority to MQSeries for AS/400 objects using the **RVKMQMAUT** command.

The Grant MQM Authority (**GRTMQMAUT**) command is used to grant specific authority for the object named in the command to another user or group of users.

Revoke MQM Authority (**RVKMQMAUT**) is used to reset, or take away previously granted authority. The names of the objects, their types, and the users and groups may all be given generically.

By authorizing *PUBLIC to an object, or set of objects, all users of the system gain that authority.

The authorizations apply to objects belonging to a particular queue manager. If you do not specify the name of a queue manager it is assumed that the default queue manager, if it exists, should be used.

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 3. Authorizations for MQI calls

AUT	Description
***ALTUSR**	Allows another user's authority to be used for MQOPEN and MQPUT1 calls.
***BROWSE**	Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
***CONNECT**	Connect the application to the specified queue manager by issuing an MQCONN call.
***GET**	Retrieve a message from a queue by issuing an MQGET call.
***INQ**	Make an inquiry on a specific queue by issuing an MQINQ call.
***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 have to be authorized for each of them.

Table 4. 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 5. 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 6. 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 could be, but does not have to 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')
```

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

Understanding the authorization specification tables

The authorization specification tables starting on page 48 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 §, 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, or queue manager.

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 may 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 may 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—which is obtained through an authorization for the queue-manager object—is required.

Table 7 on page 48 summarizes the authorizations needed for each call.

Authorization specification tables

Table 7. Security authorization needed for MQI calls

Authorization required for:	**Queue object (1)**	**Process object**	**Queue manager object**	**Namelists**
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT	Not applicable
MQOPEN Option				
MQOO_INQUIRE	MQZAO_INQUIRE **(2)**	MQZAO_INQUIRE **(2)**	MQZAO_INQUIRE **(2)**	MQZAO_INQUIRE **(2)**
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check	Not applicable
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check	Not applicable
MQOO_SAVE_ ALL_CONTEXT **(3)**	MQZAO_INPUT	Not applicable	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(4)**	MQZAO_OUTPUT	Not applicable	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(5)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check	Not applicable
MQOO_PASS_ALL_ CONTEXT **(5, 6)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check	Not applicable
MQOO_SET_ IDENTITY_CONTEXT **(5, 6)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7)**	Not applicable
MQOO_SET_ ALL_CONTEXT **(5, 8)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**	Not applicable
MQOO_OUTPUT (Transmission queue) **(9)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**	Not applicable
MQOO_SET	MQZAO_SET	Not applicable	No check	Not applicable
MQOO_ALTERNATE_ USER_AUTHORITY	**(10)**	**(10)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(10, 11)**	**(10)**
MQPUT1 Option				
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT **(12)**	Not applicable	No check	Not applicable
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT **(12)**	Not applicable	No check	Not applicable
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT **(12)**	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7)**	Not applicable
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT **(12)**	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**	Not applicable
(Transmission queue) **(9)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**	Not applicable
MQPMO_ALTERNATE_ USER_AUTHORITY	**(13)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(11)**	Not applicable
MQCLOSE Option				
MQCO_DELETE	MQZAO_DELETE **(14)**	Not applicable	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(14)**	Not applicable	Not applicable	Not applicable

Notes for Table 7:

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 of 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 specific-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 which returned the object handle being used.

    Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all of 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 8 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

Table 8. MQSC commands and security authorization needed

(2) Authorization required for:	**Queue object**	**Process object**	**Queue manager object**	**Namelists**
MQSC command				
ALTER object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
CLEAR QLOCAL	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable
DEFINE object NOREPLACE **(3)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE **(4)**
DEFINE object REPLACE **(3, 5)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
DELETE object	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE
DISPLAY object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY

Notes for Table 8:

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 does in fact already exist. 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 9 summarizes the authorizations needed for each PCF command.

Table 9. PCF commands and security authorization needed

(2) Authorization required for:	**Queue object**	**Process object**	**Queue manager object**	**Namelists**
PCF command				
Change object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
Clear Queue	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable
Copy object (without replace) **(3)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE (4)
Copy object (with replace) **(3, 6)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
Create object (without replace) **(5)**	MQZAO_CREATE **(4)**	MQZAO_CREATE **(4)**	Not applicable	MQZAO_CREATE **(4)**
Create object (with replace) **(5, 6)**	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
Delete object	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE
Inquire object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY
Inquire object names	No check	No check	No check	No check
Reset queue statistics	MQZAO_DISPLAY and MQZAO_CHANGE	Not applicable	Not applicable	Not applicable

Notes for Table 9:

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 of 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.

Authorizations for different types of object

Table 10 shows the authorities that can be given to the different object types.

Table 10. Specifying authorizations for different object types

Authority	Queue	Process	Qmgr	Namelist
all	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No
browse	Yes	No	No	No
chg	Yes	Yes	Yes	Yes
clr	Yes	No	No	No
connect	No	No	Yes	No
crt	Yes	Yes	Yes	Yes
dlt	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	Yes
put	Yes	No	No	No
inq	Yes	Yes	Yes	Yes
get	Yes	No	No	No
passall	Yes	No	No	No
passid	Yes	No	No	No
set	Yes	Yes	Yes	No
setall	Yes	No	Yes	No
setid	Yes	No	Yes	No

Authorizations for MQI calls

altusr Allows another user's authority to be used for MQOPEN and MQPUT1 calls.

browse Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

connect Connect the application to the specified queue manager by issuing an MQCONN call.

get Retrieve a message from a queue by issuing an MQGET call.

inq Make an inquiry on a specific queue by issuing an MQINQ call.

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.

Note: If you open a queue for multiple options, you have to be authorized for each of them.

Authorizations for context

passall Pass all context on the specified queue. All the context fields are copied from the original request.
passid Pass identity context on the specified queue. The identity context is the same as that of the request.
setall Set all context on the specified queue. This is used by special system utilities.
setid Set identity context on the specified queue. This is used by special system utilities.

Authorizations for commands

chg Change the attributes of the specified object.
clr Clear the specified queue (PCF Clear queue command only).
crt Create objects of the specified type.
dlt Delete the specified object.
dsp Display the attributes of the specified object.

Authorizations for generic operations

all Use all operations applicable to the object.
alladm Perform all administration operations applicable to the object.
allmqi Use all MQI calls applicable to the object.

Object Authority Manager guidelines

Some operations are particularly sensitive and should be limited to privileged users. For example,

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

You should limit the authority to create queues to privileged users. If you do not, users may 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 an MQSeries object. This is essential where a server receives requests from a program and the server wishes to ensure that the program has the required authority for the request. The server may 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 MQSeries 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 *MQSeries Application Programming Reference* manual.

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

Remote security considerations

For remote security, you should 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 the *MQSeries Intercommunication* book.

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 MQSeries 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 MQSeries for AS/400 V5.1 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 the *MQSeries Programmable System Management* manual for the PCF security requirements.

MQSC channel commands

You can issue MQSC channel commands to a remote MQSeries 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.

Chapter 6. The MQSeries 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. [1]

Queue managers, message channel agents, and applications can put messages on the DLQ. All messages on the DLQ should 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. You are strongly recommended to 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 MQSeries environments, there should be a routine that runs regularly to process messages on the DLQ. MQSeries 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/QTXTCSRC)
          MQMNAME(MY.QUEUE.MANAGER)
```

- In the rules table. For example:

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

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. Furthermore, you must be authorized to assume the identity of other users, if the DLQ handler is to be able to put messages on queues with the authority of the user ID in the message context.

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

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. Please 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***|'_')**

This parameter allows you to name the DLQ you want to process:

1. If you specify an UDLMSGQ value (or *DFT) as a parameter to the STRMQMDLQ command, this 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***|'_')**

This parameter allows you to name 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)**

This parameter is 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***)**

This parameter indicates 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 This parameter 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.

You are recommended to 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, you are recommended to reinvoke it by means of triggering.

You can supply the name of the DLQ as an input parameter of 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)

Figure 5 shows an example rule from a DLQ handler rules table.

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

Figure 5. An example rule from a DLQ handler rules table. 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. Please 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. You use these to specify values against which messages on the DLQ are matched. All pattern-matching keywords are optional.

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

APPLNAME (*PutApplName***|*****)**
This parameter is 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***|*****)**
This parameter is the *PutApplType* value specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName***|*****)**
This parameter is the name of the message queue for which the message is destined.

DESTQM (*QueueManagerName***|*)**
This parameter is 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***|*)**
This parameter is the name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType***|*)**
This parameter is 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***|*)**
This parameter is 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***|*)**
This parameter is 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***|*)**
This parameter is the reply-to queue name specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName***|*)**
This parameter is the queue manager name, of the reply-to queue, specified in the REPLYQ keyword.

USERID (*UserIdentifier***|*)**
This parameter is 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. You use these to describe how a matching message is processed.

ACTION (DISCARD|IGNORE|RETRY|FWD)
This describes 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)**
This parameter defines the name of the message queue to which the message is forwarded when you select the ACTION keyword.

QueueName
This parameter is the name of a message queue. FWDQ(' ') is not valid.

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

&REPLYQ
Takes the 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|'_')**
Identifies the queue manager of the queue to which a message is forwarded.

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

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

&REPLYQM
Takes the name from the *ReplyToQMgr* field in the message descriptor, MQMD.

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

HEADER (YES|NO)
Specifies 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)
Defines 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)**
This parameter is the number of times, in the range 1–999 999 999, that an action should be attempted (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 reasons of portability, the significant length of a line should not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (−) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

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

  results in 'ABCD'.

```
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 may 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 the *MQSeries Messages* book.) 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 may 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 could 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). Therefore, if the queue is not empty, the DLQ is periodically rescanned to check all messages. For these reasons, you should 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, the final rule in the table should be 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 are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

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

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

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

* The CCCC corporation considers itself very security
```

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

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

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

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We 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. Instrumentation events

You can use MQSeries instrumentation events to monitor the operation of queue managers. This chapter provides a short introduction to instrumentation events. For a more complete description, see the *MQSeries Programmable System Management* book.

What are instrumentation events?

Instrumentation events cause special messages, called *event messages*, to be generated whenever the queue manager detects a predefined set of conditions. For example, the following conditions give rise to a *Queue Full* event:

- Queue Full events are enabled for a specified queue, and
- An application issues an MQPUT call to put a message on that queue, but the call fails because the queue is full.

Other conditions that can give rise to instrumentation events include:

- A predefined limit for the number of messages on a queue being reached
- A queue not being serviced within a specified time
- A channel instance being started or stopped
- An application attempting to open a queue and specifying a user ID that is not authorized

With the exception of channel events, all instrumentation events must be enabled before they can be generated.

Figure 6 on page 68 summarizes the production of an event message.

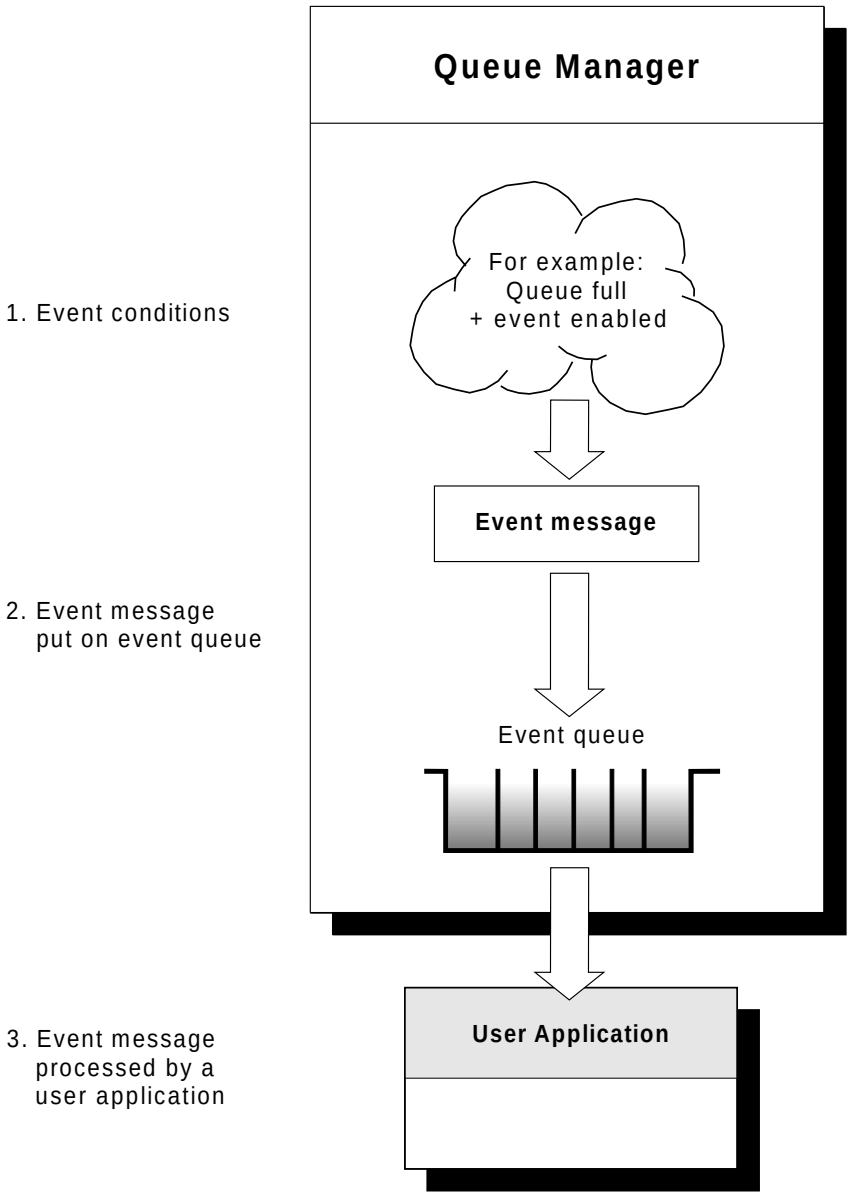

Figure 6. Understanding instrumentation events. When a queue manager detects that the conditions for an event have been met, it puts an event message on the appropriate event queue.

The event message contains information about the conditions giving rise to the event. An application can retrieve the event message from the event queue for analysis.

Why use events?

If you define your event queues as remote queues, you can put all the event queues on a single queue manager (for those nodes that support instrumentation events). You can then use the events generated to monitor a network of queue managers from a single node. Figure 7 on page 69 illustrates this.

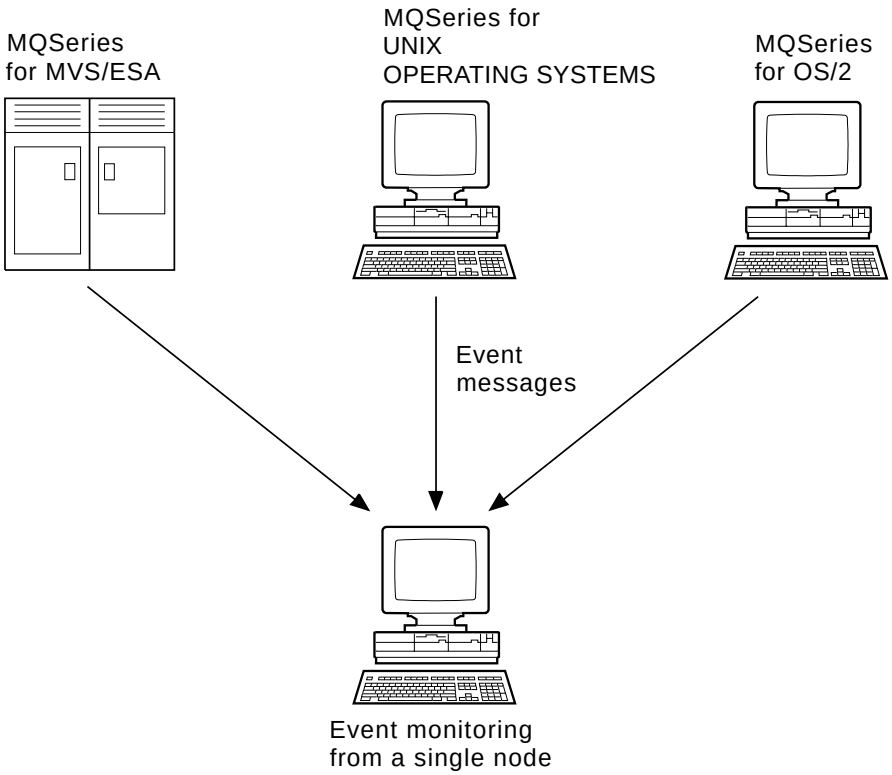

Figure 7. Monitoring queue managers across different platforms, on a single node

Types of event

MQSeries events are categorized as follows:

Queue manager events
These events are related to the definitions of resources within queue managers. For example, if an application attempts to update a resource but the associated user ID is not authorized to perform that operation, a queue manager event is generated.

Performance events
These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached or, following an MQGET request, a queue has not been serviced within a predefined period of time.

Channel events
These events are reported by channels as a result of conditions detected during their operation. For example, a channel event is generated when a channel instance is stopped.

Trigger events

When we discuss triggering in this and other MQSeries books, we sometimes refer to a *trigger event*. This occurs when a queue manager detects that the conditions for a trigger event have been met. For example, a queue can be configured to generate a trigger event each time a message arrives. (The conditions for trigger events and instrumentation events are quite different.)

A trigger event causes a trigger message to be put on an initiation queue and, optionally, an application program is started.

Event notification through event queues

When an event occurs, the queue manager puts an event message on the appropriate event queue (if such a queue has been defined). The event message contains information about the event that you can retrieve by writing a suitable MQI application program that:

- Gets the message from the queue.
- Processes the message to extract the event data. For a description of event message formats, see the *MQSeries Programmable System Management* book.

Each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

This event queue...	Contains messages from...
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events

You can define event queues as either local or remote queues. If you define all your event queues as remote queues on the same queue manager, you can centralize your monitoring activities.

Using triggered event queues

You can set up the event queues with triggers so that, when an event is generated, the event message being put onto the event queue starts a (user-written) monitoring application. This application can process the event messages and take appropriate action. For example, some events can require that an operator be informed, while others could start an application that performs some administration tasks automatically.

Enabling and disabling events

You enable and disable events by specifying the appropriate values for the queue manager, or queue attributes, or both, depending on the type of event. You do this using one of the following:

- MQSC commands. For more information, see the *MQSeries Command Reference* manual.
- PCF commands. For more information, see the *MQSeries Programmable System Management* book.
- MQAI commands. For more information, see the *MQSeries Administration Interface Programming Guide and Reference* book.

Enabling an event depends on the category of the event:

- Queue manager events are enabled by setting attributes of the queue manager.
- Performance events as a whole must be enabled on the queue manager, or no performance events can occur. You enable the specific performance events by setting the appropriate queue attribute. You also have to identify the conditions, such as a queue depth high limit, that give rise to the event,
- Channel events occur automatically; they do not need to be enabled. If you do not want to monitor channel events, you can inhibit MQPUT requests to the channel event queue.

Event messages

Event messages contain information relating to the origin of an event, including the type of event, the name of the application that caused the event and, for performance events, a short statistics summary for the queue.

The format of event messages is similar to that of PCF response messages. The message data can be retrieved from them by user-written administration programs using the data structures described in the *MQSeries Programmable System Management* book.

Chapter 8. Backup, recovery, and restart

MQSeries for AS/400 utilizes the OS/400 journaling support to aid in its backup and restore strategy. You should be familiar with standard AS/400 backup and recovery methods, and with the use of journals and their associated journal receivers on AS/400 before reading this section. For information on these topics, see the *AS/400 Backup and Recovery* book.

To understand the backup and recovery strategy, you should first understand how MQSeries for AS/400 organizes its data in the OS/400 file system and the integrated file system (IFS)

MQSeries for AS/400 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 MQSeries configuration files, the descriptions of MQSeries 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.

MQSeries for AS/400 journals

MQSeries for AS/400 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 will be detached and replaced by a new journal receiver.

The journal receivers associated with the local MQSeries for AS/400 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 will need to do both the following:
   a. Delete journals no longer needed (see "Journal management" on page 78).
   b. Record the journal changes into the latest journal receiver using (RCDMQMIMG) and then repeat the previous step. This will allow 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. More information on how the system manages receivers, see the *AS/400 Backup and Recovery* book.

The journal receiver's threshold value is 65,536 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 10. Configuring MQSeries" on page 101 for further details on configuring the system.

If you need to change the size of journal receivers after the queue manager has been created, you must 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)

The new receiver must now be attached to the AMQAJRN journal with the command:

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

See "Journal management" on page 78 for details on how to manage these journal receivers.

MQSeries for AS/400 journal usage

Persistent updates to message queues happen in two stages. The records representing the update are firstly 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, MQSeries 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 MQSeries. They are taken when the queue manager starts and shuts down, when logging space is running low, and after every 1000 operations logged.

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

When MQSeries 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 MQSeries for AS/400 uses the journal, consider the case of a local queue called `TESTQ` in the queue manager `TESTQM`. This is represented by the IFS file:

```
/QIBM/UserData/mqm/qmgrs/TESTQM/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 8.

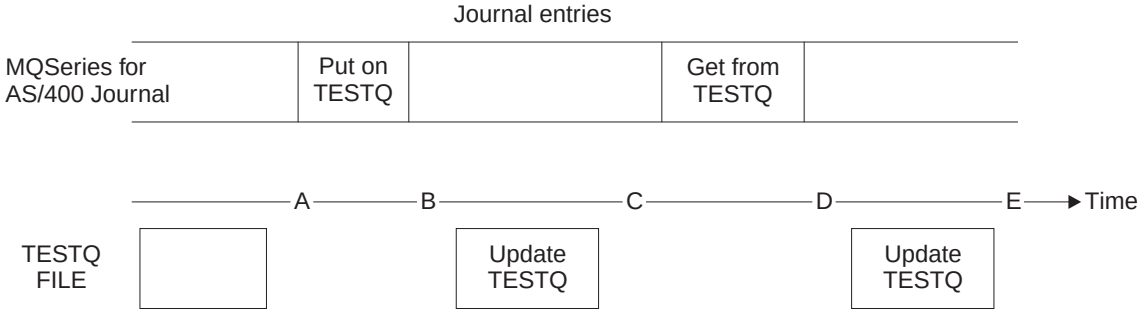

Figure 8. Sequence of events when updating MQM objects

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 MQSeries for AS/400 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 MQSeries for AS/400 to provide recovery of persistent messages after system failure. MQSeries for AS/400 remembers a particular entry in the journal receivers, and ensures that on startup it will replay the entries in the journals from this point onwards. This startup entry is periodically recalculated so that MQSeries for AS/400 only has to perform the minimum necessary replay on the next startup.

MQSeries for AS/400 provides individual recovery of objects. All persistent information relating to an object is recorded in the local MQSeries for AS/400 journals. Any MQSeries for AS/400 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 the *AS/400 Backup and Recovery* book.

Media images

For an MQSeries for AS/400 object of long duration, this can represent a large number of journal entries, going back to the point at which it was created. To avoid this overhead, MQSeries for AS/400 has the concept of a *media image* of an object.

This media image is a complete copy of the MQSeries for AS/400 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 MQSeries for AS/400 object is referred to as its *media recovery entry*.

Images of the three important MQM objects, that is, the *CTLG object, the *ADM object, and the *MQM object, are regularly taken because these objects are required for MQSeries for AS/400 to run at all.

Images of other objects are taken when convenient, particularly when the MQSeries for AS/400 queue manager is ended. MQSeries for AS/400 keeps track of the:
- Media recovery entry for each MQM object
- Oldest entry from within this set

MQSeries for AS/400 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 may never happen for some objects, for example, queues which consistently contain large numbers of messages.

Rather than allow the date of the oldest media recovery entry to continue for an unnecessarily long period, you should use the MQSeries for AS/400 command RCDMQMIMG This command enables you to take an image of selected objects manually.

Recovery from media images

MQSeries for AS/400 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 MQSeries for AS/400 command RCRMQMOBJ.

This command replays the entries in the journal to recreate the MQSeries object. Should an MQSeries object become damaged, the only valid actions that may be performed are to delete it or to re-create it by this method. Note, however, that nonpersistent messages cannot be recovered in this fashion.

Note: The authorities of the recreated object are *not* reapplied by this method. The appropriate authorities *must* be manually set after the object has been recreated. When an object is recreated, message AMQ7461 is sent to the system operator as a reminder to recreate the object authorities.

Backups of MQSeries for AS/400 data

There are two general types of MQSeries backup that should be considered:

Data and journal backup of a particular queue manager

This involves a full backup of a queue manager library and its associated IFS directories. To take a full backup of a queue manager's data, you must:

1. Use the RCDMQMIMG command to record an MQM image for all MQSeries objects.
2. End channels and ensure that the queue manager is not running. If your queue manager is running, stop it with the ENDMQM command.

   Note: If you try to take a backup of a running queue manager, the backup may not be consistent because of updates in progress when the files were copied.
3. Locate the queue manager library and IFS directories under which the queue manager places its journals and data.

   You can use the information in the configuration files to determine these directories. For further details see "Chapter 10. Configuring MQSeries" on page 101.

   Note: You may have some difficulty in understanding the names that appear in the directory. This is because the names are transformed to ensure that they are compatible with the platform on which you are using MQSeries. For more information about

name transformations, see “Understanding MQSeries queue manager library names” on page 111.

4. Back up the queue manager library by issuing the following AS/400 command, `SAVLIB LIB(QMTESTQM)`.

   A save-while-active request cannot complete unless all commitment definitions with pending changes are committed or rolled back. Therefore, if this command is used when there are active MQSeries channels, the channel connections may not end normally.

5. Back up the queue manager IFS directories by issuing the following AS/400 command:
   `SAV DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/testqm'))`

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 should be performed by issuing the following command:

```
SAVOBJ OBJ(AMQ*) LIB(QMGRLIB) OBJTYPE(*JRNRCV) .........
```

where `QMGRLIB` is the library associated with the queue manager that you are backing up.

A simple backup strategy is to perform a full backup of the MQSeries for AS/400 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, you should take care of your journal receivers. It is useful to remove journal receivers from the MQSeries for AS/400 libraries, in order to:

1. Release space – 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 9 on page 79.

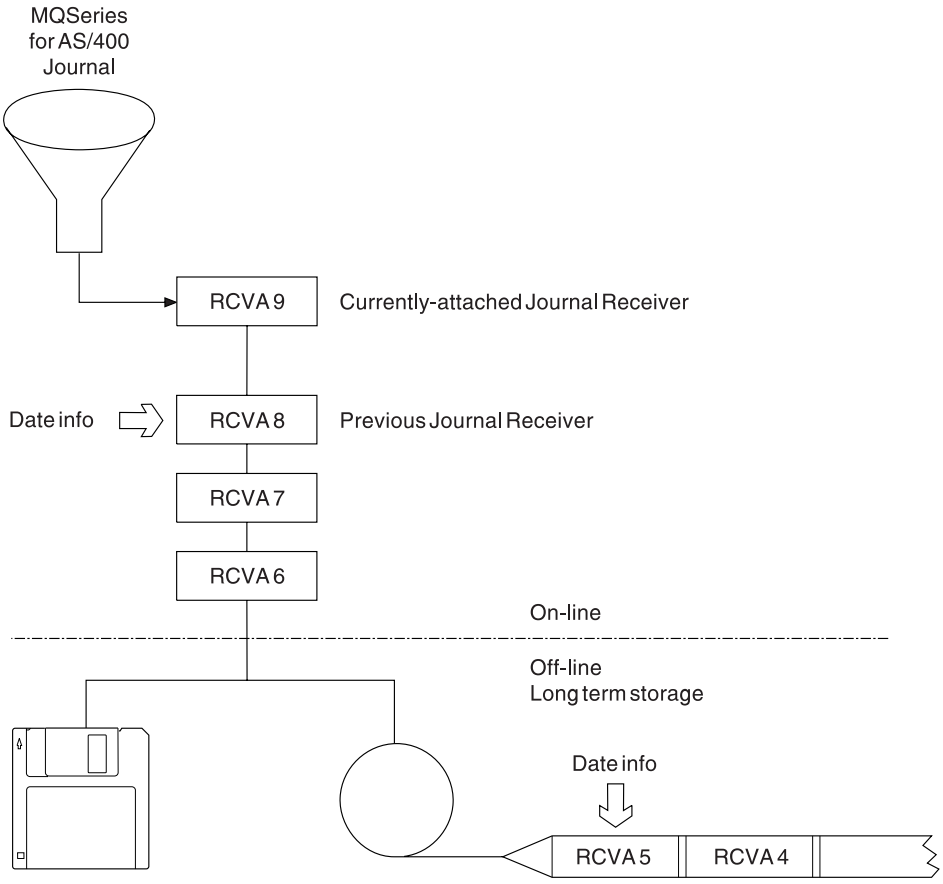

Figure 9. MQSeries for AS/400 journaling

It is important to know how far back in the journals MQSeries for AS/400 is likely to need to go, in order to determine when a journal receiver that has been backed up may be removed from the queue manager library, and when the backup itself may be discarded.

To help determine this time, MQSeries for AS/400 issues two messages to the queue manager message queue (QMQMMSG in the queue manager library) whenever it starts up, and whenever it changes a local journal receiver. These messages are:

AMQ7460

Startup recovery point. This message defines the date and time of the startup entry from which MQSeries for AS/400 replays the journal in the event of a startup recovery pass. If the journal receiver that contains this record is available in the MQSeries for AS/400 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 that may be used for recreating an object from its media image.

The journal receiver identified is the oldest one required. Any other MQSeries 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.

Note: Periodically performing RCDMQMIMG(*ALL) can save startup time for MQSeries and reduce the number of local journal receivers you need to save and restore for recovery.

MQSeries for AS/400 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, all journal receivers that were detached after this date should be restored from the backup, in order to allow the recovery pass to succeed.

The journal receiver that contains the startup entry, and any subsequent journal receivers should be kept available in the queue manager library.

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

Restoring a complete queue manager (data and journals)

If you need to recover one or more MQSeries for AS/400 queue managers from a backup you should perform the following steps.

1. Quiesce the MQSeries for AS/400 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 MQSeries for AS/400 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, the fully saved one only should be restored. You are recommended to restore journals individually, in chronological order.
4. Perform a RST operation, to restore the MQSeries 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 will replay all journal records written since the full backup and restores all the MQSeries for AS/400 objects to the consistent state at the time of the journal backup.

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 AS/400 RSTOBJ command:

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

It may be that a series of journal receivers needs to be restored, rather than simply a single receiver. For example, AMQA000007 is the oldest receiver in the MQSeries for AS/400 libraries, and both AMQA000005 and AMQA000006 need to be restored.

In this case the receivers should be restored individually in reverse chronological order. This is not always necessary, but is good practice. In severe situations, the OS/400 command WRKJRNA may be required, in order to associate the restored journal receivers with the journal.

When restoring journals, the system automatically creates an attached journal receiver with a new name in the journal receiver sequence. However, the new name generated may 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 wish to restore, which you will be unable to restore.

The solution is:

1. Manually create the next journal receiver (see "MQSeries for AS/400 journals" on page 73).
   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 - AMQA900000.

Performance considerations

If you are using a large number of persistent messages or large messages in your applications, there is an associated overhead due to the journaling of these messages.

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

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 the *OS/400 V4R4M0 Backup and Recovery* manual.

Chapter 9. Analyzing problems

This chapter suggests reasons for problems you may have with MQSeries for AS/400. This process is called problem determination. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

You should not confuse problem determination with problem solving; however, the process of problem determination often enables you to solve a problem. For example, if you find that the cause of the problem is an error in an application program, you can solve the problem by correcting the error.

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

- A performance problem may be caused by a limitation of your hardware.
- You may find that the cause of the problem is in the MQSeries for AS/400 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"
- "Problem characteristics" on page 85
- "Determining problems with MQSeries applications" on page 88
- "Obtaining diagnostic information" on page 91
- "Error logs" on page 94
- "Dead-letter queues" on page 96
- "First-failure support technology (FFST)" on page 97
- "Performance considerations" on page 98

Preliminary checks

Before you start problem determination in detail, it is worth considering the facts to see if there is an obvious cause of the problem, or an area likely, in which to start your investigation. This approach to debugging can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

The cause of your problem could be in any of the following:

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

The sections that follow raise some fundamental questions that you will 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 MQSeries application. You are recommended to check all the suggestions at each stage.

1. Has MQSeries for AS/400 run successfully before?

   Yes Proceed to Step 2.

   No It is likely you have not installed or setup MQSeries correctly.

2. Has the MQSeries application run successfully before?

   Yes Proceed to Step 3.

   No Consider the following:

   a. The application may 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 *MQSeries 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 MQSeries 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 been 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 appropriate *MQSeries Application Programming Reference* for your programming language for a description of each return code.

d. If you have applied any PTF to MQSeries for AS/400, 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 AS/400 system remain 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 may have impacted the MQSeries 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 should now be able to resolve it, possibly with the help of other books in the MQSeries library, and in the libraries of other licensed programs.

If you have not yet found the cause, you must start to look at the problem in greater detail. The following questions should be used as pointers to the problem. Answering the appropriate question, or questions, should lead you to the cause of the problem.

Can the problem be reproduced?

If the problem is reproducible, consider the conditions under which it can be reproduced:

- 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 MQSeries for AS/400 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 may 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

You should 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, the AS/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 MQSeries for AS/400 application?

If the problem only affects 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 MQSeries for AS/400 been started?

  Check that messages are reaching the transmission queue, and 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 MQSeries for AS/400 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 MQSeries V5R1

If the problem occurs on this version of MQSeries, 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 midmorning and midafternoon, and so these are the times when load-dependent problems are most likely to occur. (If your MQSeries for AS/400 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 *MQSeries Application Programming Reference* manual 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 94 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 *MQSeries Application Programming Reference* manual 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 MQSeries 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 MQSeries, 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 MQSeries applications".

Determining problems with MQSeries applications

This section discusses problems you may encounter with MQSeries 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.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH this indicates that 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 are unable to solve the problem, contact your IBM support center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues, check 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 the *MQSeries Intercommunication* book for information about how to do this.
4. Check the channel with a PING command.

See the *MQSeries Intercommunication* book for information about how to define channels.

Does the problem affect messages?

This section deals with:

- "Messages do not appear on the queue"
- "Messages contain unexpected or corrupted information" on page 90
- "Receiving unexpected messages when using distributed queues" on page 90

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?
  - Are applications able to 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.
- Are you able to 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 will set both these values to that of the message retrieved, so you may need to reset these values in order to get another message successfully.

    Also check 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 MQSeries for AS/400 has been restarted, the message will have been lost.

If you are unable to find anything wrong with the queue, and the queue manager itself is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application get started?

  If it should have been triggered, check that the correct trigger options were specified.
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- 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 may have been changed, in which case, both applications will have to be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.
- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not really intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

  If your application 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 due to data conversion.

If these checks do not enable you to solve the problem, you should check your application logic, both for the program sending the message, and for the program receiving it.

Receiving unexpected messages when using distributed queues

If your application uses distributed queues, you should also 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 MQSeries for AS/400. 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) will contain a reason or feedback code explaining why the message could not be put on to the target queue. See the *MQSeries Application Programming Reference* manual or the *MQSeries for AS/400 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 the *MQSeries Intercommunication* book for more information about distributed queuing.

Obtaining diagnostic information

This sections tells you where to find diagnostic information about MQSeries.

1. **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 MQSeries commands issued and the sequence of those commands.
2. **MQSeries Job Log:** MQSeries specific jobs, for example, the command server and channel programs, run under the MQSeries profile QMQM. If you have a problem in these areas, review these joblogs by issuing the command `WRKSPLF QMQM` to display them.
3. **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.
4. **AS/400 Message Queue:** It is useful to view messages sent to various AS/400 message queues using the DSPMSG command. Use the command `DSPMSG QSYSOPR` to check the system operator message queue, used for MQSeries journaling messages, and job completion messages in particular.
5. **Queue Manager Message Queue:** It is useful to view messages sent to a particular queue manager, called QMQMMSG and stored in the queue manager library, using the DSPMSG command.
6. **Work with Problems:** Use the WRKPRB command to display descriptions of system problems. MQSeries reports problems related to unusual usage and internal code by using this command.
7. **Error logs in the IFS:** See "Error logs" on page 94 for further information on using the error logs generated.

8. **Generation of FFSTs:** See "First-failure support technology (FFST)" on page 97 for further information on First Failure Support Technology and an example of an MQSeries for AS/400 FFST report.

Using MQSeries for AS/400 trace

Although it will be necessary to use certain traces on occasion, running the trace facility will slow your systems.

You should also consider to what destination you want your trace information sent.

Notes:

1. To run the MQSeries for AS/400 trace commands, you must have the appropriate authority.
2. Trace data is only written when trace is ended, with option *OFF

Lifetime of trace data

Trace data remains in the system until you delete it by issuing the command TRCMQM **END*.

Unless you delete it, the trace data remains until the storage monitor is ended, at which point the trace files are written out to the `QMQM` spool.

Trace usage

A trace can be obtained using TRCMQM **ON*, doing some MQ work, and TRCMQM **OFF* at the end of the MQ work being traced.

Batch jobs inherit the trace attributes of the calling program. Setting trace before you issue the command that starts a batch job, has the advantage that the batch job will be traced from its start.

Selective trace

By default, TRCMQM traces all MQSeries product components saving the maximum amount of trace data and using the *WRAP option in cases where the trace file becomes full.

You can reduce the amount of trace data being saved, thereby improving run-time performance, using the command `TRCMQM *ON` with `F4=prompt` to customize the TRCTYPE parameter.

The options available are:

***ALL** 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 only appear 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 MQSeries 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.

***DETAIL**
Activates tracing at high-detail level for flow processing trace points.

***LQMDATA**
Output data for trace points associated with internal data buffers in the local queue manager.

***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.

***PARMS**
Activates tracing at default-detail level for flow processing trace points.

***RMTDATA**
Output data for trace points associated with internal data buffers in the communications component.

***RMTFLOW**
Output data for trace points associated with processing flow in the communications component.

***SVCDATA**
Output data for trace points associated with internal data buffers in the service component.

***SVCFLOW**
Output data for trace points associated with processing flow in the service component.

***VSNDATA**
Output data for trace points associated with the version of MQSeries running.

Trace files appear in the `/QIBM/UserData/mqm/trace` directory.

Formatting trace output

To format any trace output:

- Enter the QShell
- Enter the command

  ```
  dspmqtrc [-t Format] [-h] [-o OutputFileName] InputFileName
  ```

  where:

Diagnostic information

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.

-o *output_filename*
The name of the file into which to write formatted data.

Error logs

MQSeries uses a number of error logs to capture messages concerning the operation of MQSeries 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'
```

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 usual system editor,EDTF, to view the stream files in the IFS.

Early errors

There are a number of special cases where the above error logs have not yet been established and an error occurs. MQSeries 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, due to 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 the MQSeries configuration file is readable, and the DefaultPrefix attribute of the AllQueueManagers stanza is 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 MQSeries error log

Figure 10 on page 96 shows a typical extract from an MQSeries error log.

```
 ...
08/01/97  11:41:56 AMQ8003: MQSeries queue manager started.
EXPLANATION: MQSeries queue manager janet started.
ACTION: None.
----------------------------------------------------------------------
08/01/97  11:56:52 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
----------------------------------------------------------------------
08/01/97  11:57:26 AMQ9208: Error on receive from host 'camelot
(9.20.12.34)'.
EXPLANATION: An error occurred receiving data from 'camelot
(9.20.12.34)' over TCP/IP. This may be due to a communications failure.
ACTION: Record the TCP/IP return code 232 (X'E8') and tell the
systems administrator.
----------------------------------------------------------------
08/01/97  11:57:27 AMQ9999: Channel program ended abnormally.
EXPLANATION: Channel program 'JANET' ended abnormally.
ACTION: Look at previous error messages for channel program
'JANET' in the error files to determine the cause of the failure.
----------------------------------------------------------------
08/01/97  14:28:57 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
----------------------------------------------------------------
08/02/97  15:02:49 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
----------------------------------------------------------------
08/02/97  15:02:51 AMQ9001: Channel program ended normally.
EXPLANATION: Channel program 'JANET' ended normally.
ACTION: None.
08/02/97  15:09:27 AMQ7030: Request to quiesce the queue manager
accepted. The queue manager will stop when there is no further
work for it to perform.
EXPLANATION: You have requested that the queue manager end when
there is no more work for it.  In the meantime, it will refuse
new applications that attempt to start, although it allows those
already running to complete their work.
ACTION: None.
----------------------------------------------------------------
08/02/97  15:09:32 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
 ...
```

Figure 10. Extract from an MQSeries 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 DSPMQMQ command. If the queue contains messages, you can use the WRKMQMMSG command to display a list of messages on a queue. Option 5 displays the details of the messages and option 8 displays the message data.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue.

Problems may occur if you do not associate a dead-letter queue with each queue manager. For more information about dead-letter queues, see “Chapter 6. The MQSeries dead-letter queue handler” on page 57.

First-failure support technology (FFST)

This section describes the role of first-failure support technology (FFST).

For AS/400, FFST information is recorded in a stream file in the `/QIBM/UserData/mqm/errors` directory, and in the problem database accessed using the AS/400 command **WRKPRB**.

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

FFST errors can be classified as:

- Informational

  These MQSeries errors report inconsistencies which are recoverable, or performance related. For example:

  – Message AMQ6125 with identifier X'20807412' (axE_OBJECT_MISSING) is reported by STRMQM when a journal replay recreates a missing object.

  – Message AMQ6150 (MQMRESOURCE BUSY) indicates that a job is waiting for more than six minutes for a mutex lock. This is usually because the machine is heavily loaded.

- Severe

  These MQSeries errors report internal errors which can often be resolved by quiescing and restarting the queue manager. For example:

  – Message AMQ6110 (xecSTOP_ALL) can indicate that commit processing has failed and that the unit of work needs to be rolled back when the queue manager is restarted.

- Unexpected

  These MQSeries errors report a complex variety of internal errors. If the probe identifier ends with 99, the FFST has been caused by an unmonitored OS/400 exception. These errors cannot be fully evaluated without using the associated job log.

Some typical FFST data is shown in Figure 11 on page 98.

```
--------------------------------------------------------------------------------
|                                                                              |
| MQSeries First Failure Symptom Report                                        |
| =====================================                                        |
|                                                                              |
| Date/Time         :- Tuesday January 11 13:19:12       2000                  |
| Host Name         :- test7                                                   |
| PIDS              :- 5733A38                                                 |
| LVLS              :- 510                                                     |
| Product Long Name :- MQSeries for AS400                                      |
| Vendor            :- IBM                                                     |
| Probe Id          :- XC027028                                                |
| Application Name  :- MQM                                                     |
| Component         :- xcsRequestMutexSem                                      |
| Build Date        :- Jan 10 2000 (Collector)                                 |
| File Descriptor   :- 3                                                       |
| UserID            :- 00000159                                                |
| Job Name          :- 284235/QP0ZSPWT                                         |
| Process           :- 00002729                                                |
| Thread            :- 00000001                                                |
| Major Errorcode   :-xecL_W_LONG_LOCK_WAIT                                    |
| Minor Errorcode   :- OK                                                      |
| Probe Type        :- MSGAMQ6150                                              |
| Probe Severity    :- 3                                                       |
| Probe Description :- MQM resource busy.                                      |
--------------------------------------------------------------------------------

MQM Function Stack
xcsRequestMutexSem
xcsFFST

Owning PID
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84330:2:9f 00000AAA                                ....

Owning TID

SPP:0000 :1aefQP0ZSPWT  000159    284235 :84320:3:9f                                00000001

SCB

SPP:0000 :1aefQP0ZSPWT  000159    284235 :843c0:4:9f   40000000  00000000 93E2EB80  000830F0.......lS.....0
SPP:0000 :1aefQP0ZSPWT  000159    284235 :843d0:5:9f   80000000  00000000 EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :843e0:6:9f   80000000  00000000 EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :843f0:7:9f   00000000  00000000 DFE25AE3  640B8AE0.........S!T...\
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84400:8:9f   80000000  00000000 EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84410:9:9f   00000000  00000000 DFE25AE3  64050648.........S!T....
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84420:a:9f   01000000  00000000 EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84430:b:9f   00000040  00000040 00000AAA  00000001... ... ........
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84440:c:9f   00000000  00000000 00000000  00000000................
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84450:d:9f   00000AAA  00000AA9 00000AA9  10000000.......z...z....
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84460:e:9f   00000000  00000014 00000000  00000000................
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84470:f:9f   00000000  00000000 00000000  00000000................
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84480:10:9f   00000000 00000000  00000000  00000000................
SPP:0000 :1aefQP0ZSPWT  000159    284235 :84490:11:9f   20000000 01000000  00000000  00000000................
SPP:0000 :1aefQP0ZSPWT  000159    284235 :844a0:12:9f   00000BD1 00000000  EEFE2487  FA0023B0...J.......g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :844b0:13:9f   80000000 00000000  EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :844c0:14:9f   80000000 00000000  DFE25AE3  640C131C.........S!T....
SPP:0000 :1aefQP0ZSPWT  000159    284235 :844d0:15:9f   00000001 00000000  EEFE2487  FA0023B0...........g...¬
SPP:0000 :1aefQP0ZSPWT  000159    284235 :844e0:16:9f   80000000 00000000  EEFE2487  FA0023B0...........g...¬
```

Figure 11. FFST report

The Function Stack and Trace History are used by IBM to assist in problem determination. In most cases there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Centers.

Performance considerations

This section discusses:

- General design considerations - see “Application design considerations”
- Specific performance problems - see “Specific performance problems” on page 100

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 MQSeries for AS/400 calls are discussed in the following sections.

For more information about application design, see the *MQSeries Application Programming Guide*.

Effect of message length

Although MQSeries for AS/400 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, you should send only the essential data in a message; for example, in a request to debit a bank account, the only information that may need to be passed from the client to the server application is the account number and the amount of the debit.

Effect of message persistence

Persistent messages are journaled. Journaling messages reduces the performance of your application, so you should 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 as high as 2 GB, the maximum allowed by MQSeries for AS/400.

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 may 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 AS/400 allows is 4095. However, the default is 64. MQSeries 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 MQSeries for AS/400 queue managers.

Is your application or MQSeries for AS/400 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 midmorning and midafternoon. (If your network extends across more than one time zone, peak system load might seem to you to occur at some other time.)

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, then 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 MQSeries for AS/400 is running slowly:

- If your system is slow to respond to MQSeries for AS/400 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 being run?

Chapter 10. Configuring MQSeries

This chapter explains how to change the behavior of one or more queue managers, to suit your installation's needs.

You change MQSeries configuration information by modifying the values specified on a set of configuration attributes (or parameters) that govern MQSeries.

You change this configuration information by editing the **MQSeries configuration files**.

This chapter:

- Describes the AS/400 methods for reconfiguring MQSeries in MQSeries configuration files.
- Describes the attributes you can use to modify MQSeries configuration information in "Attributes for changing MQSeries configuration information" on page 103.
- Describes the attributes you can use to modify queue manager configuration information in "Changing queue manager configuration information" on page 106.
- Provides examples of mqs.ini and qm.ini files for MQSeries for AS/400 in "Example mqs.ini and qm.ini files" on page 109.

MQSeries configuration files

You modify MQSeries configuration attributes within:

- An MQSeries configuration file (**mqs.ini**) to effect changes for MQSeries on the node as a whole. There is one mqs.ini file for each MQSeries 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 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 will not take effect until the next time the queue manager is started.

Editing configuration files

Before attempting to edit 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 MQSeries 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, you should:
- Back up the MQSeries configuration file
- Back up the new queue manager configuration file

When do you need to edit a configuration file?

You may need to edit a configuration file if, for example you:

- Lose a configuration file; recover from backup if possible.
- Need to move one or more queue managers to a new directory.
- Need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- 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 MQSeries configuration file, mqs.ini

The MQSeries configuration file, mqs.ini, contains information relevant to all the queue managers on an MQSeries 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 of them

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 the:

- `<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 MQSeries queue manager library names” on page 111 for further information.

Attributes for changing MQSeries configuration information

The following groups of attributes appear in mqs.ini:
- The AllQueueManagers stanza
- “The DefaultQueueManager stanza” on page 104
- “The ExitProperties stanza” on page 104
- “The QueueManager stanza” on page 105

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*

This attribute specifies 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, the qmgrs structure must be created. You must stop MQSeries before changing the default prefix, and restart MQSeries only after the structures have been moved to the new location and the default prefix has been changed.

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 do contain an equivalent.

Use the ConvEBCDICNewline attribute to specify the method MQSeries is to use when converting the EBCDIC NL character into ASCII format.

NL_TO_LF

Specify NL_TO_LF if you want the EBCDIC NL character (X'15') converted to the ASCII line feed character, LF (X'0A'), for all EBCDIC to ASCII conversions.

NL_TO_LF is the default.

TABLE
Specify TABLE if you want the EBCDIC NL character converted according to the conversion tables used on your platform for all EBCDIC to ASCII conversions.

Note that the effect of this type of conversion may vary from platform to platform and from language to language; while on the same platform, the behavior may vary if you use different CCSIDs.

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 11.

Table 11. 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 or 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 the CLWLMode attribute is not specified, the cluster workload exit runs in SAFE mode.

SAFE
The SAFE option specifies that the CLWL exit is to run 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
Specify FAST if you want the cluster exit to run 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. Therefore, you should 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 will fail and you run the risk of the integrity of the queue manager being compromised.

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 MQSeries queue manager library names” on page 111 for more information about name transformation.

Name=*queue_manager_name*
This attribute specifies the name of the queue manager.

Prefix=*prefix*
This attribute specifies 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*
This attribute specifies 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*
This attribute specifies the name of the library where OS/400 objects that apply

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 106
- "The Channels stanza" on page 106
- "The TCP stanza" on page 108

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 this particular queue manager needs to be configured differently from your other ones.

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
Journal receiver size.

The Channels stanza

The `Channels` stanza contains information about the channels.

MaxChannels=100|*number*
This attribute specifies the maximum number of channels allowed. The default is 100.

MaxActiveChannels=*MaxChannels_value*
This attribute specifies 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*
This attribute specifies the maximum number of initiators.

MQIBINDTYPE=FASTPATH|STANDARD
This attribute specifies the binding for applications.

FASTPATH
Channels connect using MQCONNX FASTPATH. That is, there is no agent process.

STANDARD
Channels connect using STANDARD.

AdoptNewMCA=NO|SVR|SNDR|RCVR|CLUSRCVR|ALL|FASTPATH
If MQSeries 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 termination of an existing process and the startup of a new one for a specified channel type.

If you specify the AdoptNewMCA attribute for a given channel type but the new channel fails to start because the channel is already running:

1. The new channel tries to stop the previous one by politely inviting it to end.
2. If the previous channel server does not respond to this invitation by the time the AdoptNewMCATimeout wait interval expires, the process (or the thread) for the previous channel server is killed.
3. If the previous channel server has not ended after step 2, and after the AdoptNewMCATimeout wait interval expires for a second time, MQSeries 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.

> **Note**
> The AdoptNewMCA attribute may behave in an unpredictable fashion with FASTPATH channels because of the internal design of the queue manager. Therefore exercise great caution when enabling the AdoptNewMCA attribute for FASTPATH channels.

AdoptNewMCATimeout=60|1—3600
This attribute specifies the amount of time, in seconds, that the new process should wait for the old process to end. Specify a value, in seconds, in the range 1—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, inadvertently or maliciously, shut down. At the very least check that the channel names match.

Specify one or more values, separated by commas or blanks, from the following:

QM
This means that listener process should check that the queue manager names match.

ADDRESS
This means that the listener process should check the communications address. For example, the TCP/IP address.

NAME
This means that the listener process should check that the channel names match.

ALL
You want the listener process to check for matching queue manager names, the communications address, and for matching channel names.

`AdoptNewMCACheck=NAME,ADDRESS` is the default for FAP1, FAP2, and FAP3, while `AdoptNewMCACheck=NAME,ADDRESS,QM` is the default for FAP4 and later.

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*
This attribute specifies the default port number, in decimal notation, for TCP/IP sessions. The "well known" port number for MQSeries is 1414.

KeepAlive=YES|NO
Use this attribute to 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 AS/400 is 255.

If the backlog reaches the value of 255, the TCP/IP connection is rejected and the channel will not be able to 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.

Note: Some operating systems support a larger value than the default shown. If necessary, this can be used to avoid reaching the connection limit.

Example mqs.ini and qm.ini files

Figure 12 shows an example of an mqs.ini file.

```
#********************************************************************#
#* Module Name: mqs.ini                                             *#
#* Type       : MQSeries Configuration File                         *#
#* Function   : Define MQSeries resources for the node              *#
#*                                                                  *#
#********************************************************************#
#* Notes      :                                                     *#
#* 1) This is an example MQSeries 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 12. Example of an MQSeries configuration file

Figure 13 on page 110 shows how groups of attributes might be arranged in a queue manager configuration file.

Queue manager configuration file

```
#********************************************************************#
#* Module Name: qm.ini                                              *#
#* Type       : MQSeries 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 13. Example queue manager configuration file

Notes:

1. MQSeries on the node is using 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 MQSeries configuration file is used to locate the data associated with queue managers, a nonexistent or incorrect configuration file can cause some or all MQSeries commands to fail. Also, applications cannot connect to a queue manager that is not defined in the MQSeries configuration file.

Appendix A. MQSeries names and default objects

This appendix describes the requirements for MQSeries object names, queue manager name transformations, and lists the system default objects.

MQSeries object names

The names of the following MQSeries objects can have up to 48 single-byte characters:

- Queue manager
- Queues
- Process definitions
- Namelists

The names of channels are restricted to 20 single-byte characters.

The characters that can be used for all MQSeries 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 MQSeries queue manager library names

A library is associated with each queue manager and library names can not be more than 10 characters long. However in MQSeries, 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 111 again.

   This suffix can be incremented up to 99 times to find a valid name.

Understanding MQSeries IFS directories and files

The AS/400 Integrated File System is used extensively by MQSeries to store data. For more information about the IFS see the *Integrated File System Introduction*

Each MQSeries 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 MQSeries, 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. There are limitations to the maximum length a file name can have, and 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.

The rules governing the transformation of a queue manager name, using the example of a queue manager with the name queue.manager, are as follows:

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

For example, assuming the default prefix, the queue manager name in MQSeries for AS/400 becomes /QIBM/UserData/mqm/qmgrs/queue!manager.

Object name transformation

Object names are not necessarily valid file system names, therefore, the object names may 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 MQSeries 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 MQSeries objects required for the operation of a queue manager or channel.
- The default objects define all of 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 12 lists the system and default queue objects.
- Table 13 lists the system and default channel objects.
- Table 14 lists the system and default process objects.

Table 12. 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.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue which holds the synchronization data for channels.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered message queue).
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.

Default objects

Table 13. System and default objects - channels

Object name	Description
SYSTEM.AUTO.RECEIVER	Dynamic receiver channel
SYSTEM.AUTO.SVRCONN	Dynamic server-connection channel
SYSTEM.DEF.CLUSRCVR	Default receiver channel for the cluster used to supply default values for any attributes not specified when a CLUSRCVR channel is created on a queue manager in the cluster.
SYSTEM.DEF.CLUSSDR	Default sender channel for the cluster used to supply default values for any attributes not specified when a CLUSSDR channel is created on a queue manager in the cluster.
SYSTEM.DEF.RECEIVER	Default receiver channel.
SYSTEM.DEF.REQUESTER	Default requester channel.
SYSTEM.DEF.SENDER	Default sender channel.
SYSTEM.DEF.SERVER	Default server channel.
SYSTEM.DEF.SVRCONN	Default server-connection channel.

Table 14. System and default objects - processes

Object name	Description
SYSTEM.DEFAULT.PROCESS	Default process definition.

Appendix B. Sample resource definitions

This appendix contains the AMQSAMP4 sample AS/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               */
/*                                                                   */
/*                5763-MQ2                                           */
/*                (C) Copyright IBM Corporation 1993, 1996.          */
/*                                                                   */
/* Status: Version 3 Release 2.0                                     */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* 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('AMQ1INQ4')    /* RPG - OPM */        +
              /*        APPID('AMQ2INQ4')    /* 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('AMQ1SET4')    /* RPG - OPM */        +
              /*        APPID('AMQ2SET4')    /* 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('AMQ1ECH4')    /* RPG - OPM */        +
              /*        APPID('AMQ2ECH4')    /* RPG - ILE */

/********************************************************************/
/*   Normal return.                                                 */
/********************************************************************/
            RETURN
            ENDPGM

/********************************************************************/
/* END OF AMQSAMP4                                                  */
/********************************************************************/
```

AMQSAMP4

Appendix C. 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.

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.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

AIX	Application System/400	AS/400
BookManager	C/400	CICS
COBOL/400	FFST	First Failure Support Technology
IBM	IMS	MQSeries
MQSeries Three Tier	MVS/ESA	OS/2
OS/400	RACF	VisualAge

Lotus and LotusScript are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Intel is a trademark of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, may be the trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

administration bag. In the MQAI, a type of data bag that is created for administering MQSeries by implying that it can change the order of data items, create lists, and check selectors within a message.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

Advanced Program-to-Program Communication (APPC). The general facility characterizing the LU 6.2 architecture and its various implementations in products.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

alias queue object. An MQSeries object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

alternate user security. A security feature in which the authority of one user ID can be used by another user ID; for example, to open an MQSeries object.

APAR. Authorized program analysis report.

APPC. Advanced Program-to-Program Communication.

application log. In Windows NT, a log that records significant application events.

application queue. A queue used by an application.

asynchronous messaging. A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute. One of a set of properties that defines the characteristics of an MQSeries object.

authorization checks. Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

authorization file. In MQSeries on UNIX systems, a file that provides security definitions for an object, a class of objects, or all classes of objects.

authorization service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

backout. An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

bag. See *data bag*.

browse. In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor. In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel control function (CCF). In MQSeries, a program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. A time when significant information is written on the log. Contrast with *syncpoint*. In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

CICS transaction. In CICS, a unit of application processing, usually comprising one or more units of work.

circular logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping all restart data in a ring of log files. Logging fills the first file in the ring and then moves on to the next, until all the files are full. At this point, logging goes back to the first file in the ring and starts again, if the space has been freed or is no longer needed. Circular logging is used during restart recovery, using the log to roll back transactions that were in progress when the system stopped. Contrast with *linear logging*.

client. A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQSeries client*.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client connection channel type. The type of MQI channel definition associated with an MQSeries client. See also *server connection channel type*.

cluster. A network of queue managers that are logically associated in some way.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an administration instruction that can be carried out by the queue manager.

command bag. In the MQAI, a type of bag that is created for administering MQSeries objects, but cannot change the order of data items nor create lists within a message.

command processor. The MQSeries component that processes commands.

command server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code. A return code indicating how an MQI call has ended.

configuration file. In MQSeries on UNIX systems, MQSeries for AS/400, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file that contains configuration information related to, for example, logs, communications, or installable services. Synonymous with *.ini file*. See also *stanza*.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN call, or automatically by the MQOPEN call.

connection handle. The identifier or token by which a program accesses the queue manager to which it is connected.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control command. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. Such a command requires only that the MQSeries product be installed; it does not require a special utility or program to run it.

controlled shutdown. See *quiesced shutdown*.

D

data bag. In the MQAI, a bag that allows you to handle properties (or parameters) of objects.

data item. In the MQAI, an item contained within a data bag. This can be an integer item or a character-string item, and a user item or a system item.

data conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCE. Distributed Computing Environment.

DCI. Data conversion interface.

dead-letter queue (DLQ). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

default object. A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

distributed application. In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Distributed Computing Environment (DCE). Middleware that provides some basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue.*

E

event. See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

event data. In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

event header. In an event message, the part of the message data that identifies the event type of the reason code for the event.

event log. See *application log.*

event message. Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

event queue. The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

Event Viewer. A tool provided by Windows NT to examine and manage log files.

F

FFST. First Failure Support Technology.

FIFO. First-in-first-out.

First Failure Support Technology (FFST). Used by MQSeries on UNIX systems, MQSeries for OS/2 Warp, MQSeries for Windows NT, and MQSeries for AS/400 to detect and report software problems.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

Framework. In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:

- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

G

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse.*

H

handle. See *connection handle* and *object handle.*

I

ILE. Integrated Language Environment.

immediate shutdown. In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

Integrated Language Environment (ILE). The AS/400 Integrated Language Environment. This replaces the AS/400 Original Program Model (OPM).

.ini file. See *configuration file*.

initiation queue. A local queue on which the queue manager puts trigger messages.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

installable services. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

instrumentation event. A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

Internet Protocol (IP). A protocol used to route data from its source to its destination in an Internet environment. This is the base layer, on which other protocol layers, such as TCP and UDP are built.

IP. Internet Protocol.

L

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

listener. In MQSeries distributed queuing, a program that monitors for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

locale. On UNIX systems, a subset of a user's environment that defines conventions for a specific culture (such as time, numeric, or monetary formatting and character classification, collation, or conversion). The queue manager CCSID is derived from the locale of the user ID that created the queue manager.

local queue. A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

log control file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file in which all significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, MQSeries allocates secondary log files.

logical unit of work (LUW). See *unit of work*.

LU 6.2. A type of logical unit (LU) that supports general communication between programs in a distributed processing environment.

M

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows

NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. In system programming, information intended for the terminal operator or system administrator.

message channel. In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

message channel interface (MCI). The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

message descriptor. Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

message priority. In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue. Synonym for *queue*.

message queue interface (MQI). The programming interface provided by the MQSeries queue managers. This programming interface allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

messaging. See *synchronous messaging* and *asynchronous messaging*.

model queue object. A set of queue attributes that act as a template when a program creates a dynamic queue.

MQAI. MQSeries Administration Interface.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries Administration Interface (MQAI). A programming interface to MQSeries.

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQSeries commands (MQSC). Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects. Contrast with *programmable command format (PCF)*.

N

namelist. An MQSeries object that contains a list of names, for example, queue names.

name service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the facility that determines which queue manager owns a specified queue.

name service interface (NSI). The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

name transformation. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, an internal process that changes a queue manager name so that it is unique and valid for the system being used. Externally, the queue manager name remains unchanged.

NetBIOS. Network Basic Input/Output System. An operating system interface for application programs used on IBM personal computers that are attached to the IBM Token-Ring Network.

New Technology File System (NTFS). A Windows NT recoverable file system that provides security for files.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

NTFS. New Technology File System.

null character. The character that is represented by X'00'.

O

OAM. Object authority manager.

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist, or a storage class (OS/390 only).

object authority manager (OAM). In MQSeries on UNIX systems, MQSeries for AS/400, and MQSeries for Windows NT, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

object descriptor. A data structure that identifies a particular MQSeries object. Included in the descriptor are the name of the object and the object type.

object handle. The identifier or token by which a program accesses the MQSeries object with which it is working.

OPM. Original Program Model.

Original Program Model (OPM). The AS/400 Original Program Model. This is no longer supported on MQSeries. It is replaced by the Integrated Language Environment (ILE).

OTMA. Open Transaction Manager Access.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

PCF. Programmable command format.

PCF command. See *programmable command format*.

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation. In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event. A category of event indicating that a limit condition has occurred.

performance trace. An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue. A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

persistent message. A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

ping. In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

preemptive shutdown. In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

principal. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a term used for a user identifier. Used by the object authority manager for checking authorizations to system resources.

process definition object. An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

programmable command format (PCF). A type of MQSeries message used by:

- User administration applications, to put PCF commands onto the system command input queue of a specified queue manager
- User administration applications, to get the results of a PCF command from a specified queue manager
- A queue manager, as a notification that an event has occurred

Contrast with *MQSC*.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF. Program temporary fix.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager. A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. An MQSeries object that defines the attributes of a particular queue manager.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queuing. See *message queuing*.

quiesced shutdown. In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. A type of shutdown of the CICS adapter where the adapter disconnects from MQSeries, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing. In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

Registry. In Windows NT, a secure database that provides a single source for system and application configuration data.

Registry Editor. In Windows NT, the program item that allows the user to edit the Registry.

Registry Hive. In Windows NT, the structure of the data stored in the Registry.

remote queue. A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To a program, a queue manager that is not the one to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message. A type of message used for replies to request messages. Contrast with *request message* and *report message*.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. Contrast with *reply message* and *request message*.

repository. A collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, what queues they host, and so on.

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

request message. A type of message used to request a reply from another program. Contrast with *reply message* and *report message*.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

responder. In distributed queuing, a program that replies to network connection requests from another system.

resynch. In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

security enabling interface (SEI). The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

SEI. Security enabling interface.

sender channel. In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server. (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel. In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

server connection channel type. The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

service interval. A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event. An event related to the service interval.

shell. In the AIX and UNIX environments, a software interface between a user and the operating system of a computer. Shell programs interpret commands and communicate them to the operating system.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

single-phase backout. A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

SNA. Systems Network Architecture.

SPX. Sequenced Packet Exchange transmission protocol.

stanza. A group of lines in a configuration file that assigns a value to a parameter modifying the behavior of a queue manager, client, or channel. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a configuration (.ini) file may contain a number of stanzas.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

symptom string. Diagnostic information displayed in a structured format designed for searching the IBM software support database.

synchronous messaging. A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

system bag. A type of data bag that is created by the MQAI.

system.command.input queue. A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

system control commands. Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

TCP. Transmission Control Protocol.

TCP/IP. Transmission Control Protocol/Internet Protocol.

temporary dynamic queue. A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

tranid. See *transaction identifier*.

transaction. See *unit of work* and *CICS transaction*.

transaction identifier. In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

Transmission Control Protocol (TCP). Part of the TCP/IP protocol suite. A host-to-host protocol between hosts in packet-switched communications networks. TCP provides connection-oriented data stream delivery. Delivery is reliable and orderly.

Transmission Control Protocol/Internet Protocol (TCP/IP). A suite of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

triggering. In MQSeries, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI). The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

UDP. User Datagram Protocol.

UIS. User identifier service.

undelivered-message queue. See *dead-letter queue*.

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

unit of recovery. A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

user bag. In the MQAI, a type of data bag that is created by the user.

User Datagram Protocol (UDP). Part of the TCP/IP protocol suite. A packet-level protocol built directly on the Internet Protocol layer. UDP is a connectionless and less reliable alternative to TCP. It is used for application-to-application programs between TCP/IP host systems.

user identifier service (UIS). In MQSeries for OS/2 Warp, the facility that allows MQI applications to associate a user ID, other than the default user ID, with MQSeries messages.

utility. In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

X

X/Open XA. The X/Open Distributed Transaction Processing XA interface. A proposed standard for distributed transaction communication. The standard specifies a bidirectional interface between resource managers that provide access to shared resources

within transactions, and between a transaction service that monitors and resolves transactions.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries "family" books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX), V2.2.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V2.1
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT, V5.1

Any exceptions to this general rule are indicated.

MQSeries Brochure

The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing

An Introduction to Messaging and Queuing, GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide

The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication

The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Queue Manager Clusters

MQSeries Queue Manager Clusters, SC34-5349, describes MQSeries clustering. It explains the concepts and terminology and shows how you can benefit by taking advantage of clustering. It details changes to the MQI, and summarizes the syntax of new and changed MQSeries commands. It shows a number of examples of tasks you can perform to set up and maintain clusters of queue managers.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Clients

The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration

The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem

determination, and the dead-letter queue handler. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:
- MQSeries for AIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1

MQSeries Command Reference

The *MQSeries Command Reference*, SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management

The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, Programmable Command Format (PCF) messages, and installable services.

MQSeries Administration Interface Programming Guide and Reference

The *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390, provides information for users of the MQAI. The MQAI is a programming interface that simplifies the way in which applications manipulate Programmable Command Format (PCF) messages and their associated data structures.

This book applies to the following MQSeries products only:
- MQSeries for AIX V5.1
- MQSeries for AS/400 V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Messages

The *MQSeries Messages* book, GC33-1876, which describes "AMQ" messages issued by MQSeries, applies to these MQSeries products only:
- MQSeries for AIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

For other MQSeries platforms, the messages are supplied with the system. They do not appear in softcopy manual form.

MQSeries Application Programming Guide

The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference

The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary

The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference* manual.

MQSeries Using C++

MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by these MQSeries products:
- MQSeries for AIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for OS/390, V2.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1

MQSeries C++ is also supported by MQSeries clients supplied with these products and installed in the following environments:
- AIX
- HP-UX

- OS/2
- Sun Solaris
- Windows NT
- Windows 3.1
- Windows 95 and Windows 98

MQSeries Using Java

MQSeries Using Java, SC34-5456, provides both guidance and reference information for users of the MQSeries Bindings for Java and the MQSeries Client for Java. MQSeries classes for Java are supported by these MQSeries products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for MVS/ESA V1.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1

This book is available in softcopy only.

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX

MQSeries for AIX, V5.1 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 V5.1 Quick Beginnings, GC34-5557

MQSeries for AS/400 V5.1 System Administration, SC34-5558

MQSeries for AS/400 V5.1 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX

MQSeries for AT&T GIS UNIX System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS

MQSeries for Digital OpenVMS System Management Guide, GC33-1791

MQSeries for Digital UNIX (Compaq Tru64 UNIX)

MQSeries for Digital UNIX System Management Guide, GC34-5483

MQSeries for HP-UX

MQSeries for HP-UX, V5.1 Quick Beginnings, GC33-1869

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp, V5.1 Quick Beginnings, GC33-1868

MQSeries for OS/390

MQSeries for OS/390 Version 2 Release 1 Licensed Program Specifications, GC34-5377

MQSeries for OS/390 Version 2 Release 1 Program Directory

MQSeries for OS/390 System Management Guide, SC34-5374

MQSeries for OS/390 Messages and Codes, GC34-5375

MQSeries for OS/390 Problem Determination Guide, GC34-5376

MQSeries link for R/3

MQSeries link for R/3 Version 1.2 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for Sun Solaris

MQSeries for Sun Solaris, V5.1 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA

MQSeries for VSE/ESA Version 2 Release 1 Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries for Windows

MQSeries for Windows V2.0 User's Guide, GC33-1822

MQSeries for Windows V2.1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT, V5.1 Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

BookManager format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

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

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

`http://www.adobe.com/`

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

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

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at:

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

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Related publications

AS/400 CL Reference Common CL Information, SC41-5722
AS/400 Backup and Recovery, SC41-5304
AS/400 Security - Reference, SC41-5302
AS/400 National Language Support, SC41-5101
AS/400 System API Reference, SC41-5801

Index

A

ACTION keyword, rules table 60
action keywords, rules table 60
administration
  authorizations 50
  description of 32
  introduction to 29
  local, definition of 29
  MQAI, using 32
  MQSeries (MQSC) commands 30
  PCF commands 31
  queue manager name transformation 111
  remote administration, definition of 29
  understanding MQSeries file names 111
  using PCF commands 30
alias queues
  authorizations to 53
  defining alias queues 23
  working with alias queues 23
AllQueueManagers stanza, mqs.ini 103
alternate-user authority 54
application design, performance considerations 98
application programming errors, examples of 84
application programs
  receiving messages 2
  retrieving messages from queues 3
  sending messages 2
  time-independent applications 1
application queues
  creating and copying, restrict access to 53
  defining application queues for triggering 25
  description of 6
APPLIDAT keyword, rules table 59
APPLNAME keyword, rules table 59
APPLTYPE keyword, rules table 59
attributes
  changing local queue attributes 22
  queue manager 20
  queues 5
authority
  alternate-user 54
  context authority 54
authorizations
  administration 50
  for object types 52
  MQI 47
  specification tables 47

B

backups of data 77
bibliography 133
BookManager 136

C

changing
  local queue attributes 22
  queue manager attributes 20
channels
  channel command security 55
  Channels stanza, qm.ini 106
  command security requirements 55
  description of 8
  escape command authorizations 50
  events 69
  exits 10
  security, MQSC channel commands 55
  security requirements for PCF commands 55
Channels stanza, qm.ini 106
characters allowed in object names 111
CL commands
  creating a queue
    alias 20
    initiation 20
    model 20
    remote 18
    transmission 19
    using CRTMQMQ for local queues 16
    using WRKMQMQ for local queues 17
  creating MQSeries objects 15, 99
  starting a local queue manager 15
clearing a local queue 22
clients and servers
  definitions 9
clusters
  cluster transmission queues 7
  description of 8
  ExitProperties stanza attributes 104
command files 30
command queues
  command server status 34
  description of 7
  SYSTEM.ADMIN.COMMAND.QUEUE 7
  status of command server 34
command server
  displaying status 34
  remote administration 34
  starting a command server 34
  stopping a command server 35
commands, PCF 31
configuration files
  AllQueueManagers stanza, mqs.ini 103
  Channels stanza, qm.ini 106
  DefaultQueueManager stanza, mqs.ini 104
  editing 101
  example mqs.ini file 109
  example qm.ini file 109
  ExitProperties stanza, mqs.ini 104
  Log stanza, qm.ini 106
  mqs.ini, description of 102
configuration files *(continued)*
  priorities 102
  queue manager configuration file, qm.ini 102
  QueueManager stanza, mqs.ini 105
  TCP stanza, qm.ini 108
configuring logs 106
context authority 54
CorrelId, performance considerations 99
creating
  a dynamic (temporary) queue 3
  a model queue 3
  a predefined (permanent) queue 3
  a process definition 26

D

data conversion
  ConvEBCDICNewline attribute, AllQueueManagers stanza 103
  EBCDIC NL character conversion to ASCII 103
dead-letter header, MQDLH 57
dead-letter queues
  defining a dead-letter queue 21
  description of 7
default objects 9, 113
DefaultQueueManager stanza, mqs.ini 104
defining
  a model queue 24
  an alias queue 23
  an initiation queue 26
  MQSeries queues 5
deleting a local queue 23
DESTQ keyword, rules table 59
DESTQM keyword, rules table 60
diagnostic information, obtaining 91
directories, queue manager 53
display
  default object attributes 22
  process definitions 26
distributed queuing example 27
DLQ handler
  invoking 57
  rules table 58
dynamic queues
  authorizations 53
  description of 3

E

EBCDIC NL character conversion to ASCII 103
environment variables
  MQSPREFIX 103
error logs
  errors occurring before log established 95

error logs *(continued)*
example, MQSeries 95
log files 94
escape PCFs 31
event queues
description of 8
event notification through event queue 70
triggered event queues 70
events
channel 69
enabling and disabling 70
event messages 71
event notification 70
instrumentation 67
performance events 69
queue manager events 69
trigger 70
types of 69
examples
creating a transmission queue 19
creating an alias queue 20
creating local queues
using the CRTMQMQ command 16
using the WRKMQMQ command 17
creating remote queues
as a queue manager alias 18
as a remote queue definition 18
as an alias to a reply-to queue 19
error log, MQSeries 95
mqs.ini file 109
qm.ini file 109
ExitProperties stanza,mqs.ini 104
extending queue manager facilities 10

F

FEEDBACK keyword, rules table 60
FFST (first-failure support technology) 97
file names 111
files
IFS directories 112
log files, in problem determination 94
MQSeries configuration 102
queue manager configuration 102
understanding names 111
FORMAT keyword, rules table 60
formatting trace 93
FWDQ keyword, rules table 61
FWDQM keyword, rules table 61

G

glossary 123

H

HEADER keyword, rules table 61
HTML (Hypertext Markup Language) 136
Hypertext Markup Language (HTML) 136

I

initiation queues
defining 26
initiation queues *(continued)*
description of 6
INPUTQ keyword, rules table 58
INPUTQM keyword, rules table 58
instrumentation events
description 67
event messages 71
types of 69
why use them 68

J

journal management 78
journal usage 75
journals 73

L

length of object names 111
local administration, definition of 29
local queues 20
changing queue attributes, commands to use 22
clearing 22
copying a local queue definition 22
defining 20
defining application queues for triggering 25
deleting 23
description of 5
specific queues used by MQSeries 6
working with local queues 20
Log stanza, qm.ini 106
logical unit of work, definition of 11
logs
configuring 106
errors occurring before error log established 95
log files, in problem determination 94
Log stanza, qm.ini 106

M

managing objects for triggering 25
maximum line length, MQSC commands 30
media images 76
message-driven processing 1
message length, decreasing 22
message persistence, performance considerations 99
message queuing 1
messages
application data 2
containing unexpected information 90
definition of 2
event messages 71
message descriptor 2
message-driven processing 1
message lengths 2
not appearing on queues 89
operator messages 95
queuing 1
retrieval algorithms 3
retrieving messages from queues 3
messages *(continued)*
sending and receiving 2
undelivered 96
model queues
creating a model queue 3
defining 24
working with 24
monitoring queue managers 68
MQAI, description of 31
MQDLH, dead-letter header 57
MQI (message-queuing interface)
authorization specification tables 47
authorizations 47
definition of 1
queue manager calls 5
receiving messages 2
sending messages 2
MQI authorizations 47
MQOPEN authorizations 47
MQPUT and MQPUT1, performance considerations 99
MQPUT authorizations 47
mqs.ini configuration file
AllQueueManagers stanza 103
DefaultQueueManager stanza 104
definition of 101
editing 101
ExitProperties stanza 104
priorities 102
QueueManager stanza 105
MQSC commands
authorization 50
command files, input 30
escape PCFs 31
maximum line length 30
object attribute names 4
overview 30
security requirements, channel commands 55
MQSeries Explorer
description of 32
prerequisite software 33
required resource definitions 33
MQSeries for AS/400
backups of data 77
CL commands 13
journal management 78
journal usage 75
journals 73
media images 76
recovery from media images 77
restoring a complete queue manager 80
restoring journal receivers 80
MQSeries publications 133
MQSPREFIX, environment variable 103
MQZAO, constants and authority 48
MsgId, performance considerations 99
MSGTYPE keyword, rules table 60

N

namelists, description of 9
naming conventions 3
national language support
EBCDIC NL character conversion to ASCII 103
operator messages 95

NL character, EBCDIC conversion to ASCII 103
notification of events 70

O

OAM (Object Authority Manager)
  description of 44
  guidelines for using 53
  resources protected by 44
  sensitive operations 53
object names 4
objects
  access to 43
  administration of 29
  attributes of 4
  automation of administration tasks 30
  default object attributes, displaying 22
  description of 8
  local queues 5
  managing objects for triggering 25
  multiple queues 5
  naming conventions 4
  process definitions 8
  queue manager objects used by MQI calls 5
  queue managers 4
  queue objects, using 6
  remote queues 5
  system default objects 9
  using MQSC commands to administer 30
operator
  commands, no response from 87
  messages 95

P

pattern-matching keywords, rules table 59
PCF (programmable command format)
  administration tasks 30
  attributes in PCFs 31
  authorization specification tables 47
  automating administrative tasks using PCF 31
  channel security, requirements 55
  escape PCFs 31
  MQAI, using to simplify use of 32
  object attribute names 4
PDF (Portable Document Format) 136
performance considerations
  application design 98
  CorrelId 99
  message persistence 99
  MQPUT and MQPUT1 99
  MsgId 99
  syncpoint 99
  trace 92
  variable length 99
performance events 69
permanent (predefined) queues 3
PERSIST keyword, rules table 60
Portable Document Format (PDF) 136
PostScript format 136
predefined (permanent) queues 3
problem determination
  command errors 86
  log files 94
  no response from commands 87
  preliminary checks
    operating system 87
    problem affects all users 86
    problem intermittent 86
    problem occurs at specific times 87
    problem reproducible 85
  programming errors 84
  trace 92
  undelivered messages 96
process definitions
  creating 26
  description of 8
  displaying 26
processing, message-driven 1
programming errors, examples of 84
protected resources 44
publications
  MQSeries 133
  related 136
PUTAUT keyword, rules table 61

Q

qm.ini configuration file
  Channels stanza 106
  definition of 102
  editing 101
  Log stanza 106
  priorities 102
  TCP stanza 108
queue managers
  attributes, changing 20
  authorizations 53
  command server 34
  description of 4
  directories 53
  extending queue manager facilities 10
  monitoring 68
  name transformation 111
  object authority manager, description 44
  objects used in MQI calls 5
  qm.ini files 102
  queue manager events 69
QueueManager stanza, mqs.ini 105
queues
  alias 23
  application queues 25
  attributes 5
  authorizations to 53
  changing queue attributes 22
  clearing local queues 22
  dead-letter, defining 21
  defining MQSeries queues 5
  definition of 2
  deleting a local queue 23
  dynamic (temporary) queues 3
  extending queue manager facilities 10
  initiation queues 26
  local, working with 20
queues *(continued)*
  local queues 5
  model queues 3, 24
  multiple queues 5
  predefined (permanent) queues 3
  queue managers, description of 4
  queue objects, using 6
  retrieving messages from 3
  specific local queues used by MQSeries 6

R

REASON keyword, rules table 60
recovery from media images 77
remote administration
  command server 34
  definition of remote administration 29
remote queues
  authorizations to 53
  examples of creating 18
  security considerations 54
reply-to queues,.description of 7
REPLYQ keyword, rules table 60
REPLYQM keyword, rules table 60
resources, updating under syncpoint control 10
restoring a complete queue manager 80
restoring journal receivers 80
restricting access to MQM objects 43
retrieval algorithms for messages 3
RETRY keyword, rules table 62
RETRYINT keyword, rules table 58
rules table, DLQ handler
  control data entry
    INPUTQ keyword 58
    INPUTQM keyword 58
    RETRYINT keyword 58
    WAIT keyword 58
  example of 65
  patterns and actions (rules) 59
    ACTION keyword 60
    APPLIDAT keyword 59
    APPLNAME keyword 59
    APPLTYPE keyword 59
    DESTQ keyword 59
    DESTQM keyword 60
    FEEDBACK keyword 60
    FORMAT keyword 60
    FWDQ keyword 61
    FWDQM keyword 61
    HEADER keyword 61
    MSGTYPE keyword 60
    PERSIST keyword 60
    PUTAUT keyword 61
    REASON keyword 60
    REPLYQ keyword 60
    REPLYQM keyword 60
    RETRY keyword 62
    USERID keyword 60
  processing 63
  syntax 62

S

security
  administration authorizations 50
  command security requirements 55
  considerations 43
  context authority 54
  MQI authorizations 47
  MQSC channel commands 55
  MQSeries authorities 44
  object authority manager (OAM) 44
  remote queues 54
  resources protected by the OAM 44
  security requirements for PCF commands 55
  sensitive operations, OAM 53
sensitive operations, OAM 53
servers 9
softcopy books 136
stanzas
  AllQueueManagers, mqs.ini 103
  Channels, qm.ini 106
  DefaultQueueManager, mqs.ini 104
  ExitProperties, mqs.ini 104
  Log, qm.ini 106
  QueueManager, mqs.ini 105
  TCP, qm.ini 108
starting a command server 34
stopping a command server 35
storage problems 100
STRMQMDLQ command 57
syncpoint, performance considerations 99
system default objects 9
system objects 113

T

TCP stanza, qm.ini 108
temporary (dynamic) queues 3
terminology used in this book 123
time-independent applications 1
trace, performance considerations 92
trace data
  formatting 93
  lifetime of 92
  selective 92
  usage of 92
transactional support, updating under syncpoint control 10
transmission queues
  cluster transmission queues 7
  description of 7
triggering
  defining an application queue for triggering 25
  event queues 70
  managing objects for triggering 25
  message-driven processing 1
  trigger events 70
types of event 69

U

user exits
  channel exits 10
  data conversion exits 10
USERID keyword, rules table 60
using CL commands 13

V

variable length, performance considerations 99

W

WAIT keyword, rules table 58
Windows Help 136
work management
  objects 38
  tasks 38
  using 39

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 request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should 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:

  Information Development Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–870229
  - From within the U.K., use 01962–870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5558-00

Spine information:

MQSeries® for AS/400®

MQSeries for AS/400, V5.1 System Administration

V5.1