Willow Technology

MQSeries for IRIX
System Management Guide

Version 2

First Edition – May, 1999

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

First edition (May, 1999)

This edition applies to MQSeries for IRIX Version 2 and to any subsequent releases and modifications until otherwise indicated in new editions.

If you want to make comments regarding this publication please send them to one of the folowing:

Willow Technology, Inc.
Publications
P.O. Box 320005
Los Gatos, CA 95032-0100
USA

or to email address: support@willowtech.com.

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

Table of Contents

Figures

Notices

References in this publication to Willow Technology products, programs, or services do not imply that Willow Technology intends to make these available in all countries in which Willow Technology operates.

Any reference to an Willow Technology product, program, or service is not intended to state or imply that only that Willow Technology product, program, or service may be used.

The following paragraph does not apply to any country where such provisions are inconsistent with local law:

WILLOW TECHNOLOGY, INC PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF 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.

Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of Willow Technology and its Licensors may be used instead of the Willow Technology product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Willow Technology and its Licensors, are the responsibility of the user.

Willow Technology and its Licensors may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

Trademarks

Willow Technology, the Willow logo and willowtech.com are trademarks of Willow Technology, Inc.

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

AS/400	IBM
AIX	AIX/6000
MQSeries	MVS/ESA
NetView	CICS
OS/2	OS/400
Operating System/2	PS/2
RISC System/6000	SAA

Windows is a trademark of Microsoft Corporation.

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

SGI and IRIXare registered trademarks of Silicon Graphics, Inc.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About this book

This book refers to the MQSeries for IRIX Version 2 product produced by Willow Technology.

MQSeries for IRIX is part of the MQSeries family of products. These products provide application programming services that enable application programs to communicate with each other using *message queues.* This form of communication is referred to as *commercial messaging.* The applications involved can exist on different nodes on a wide variety of machine and operating system types. They use a common application programming interface, called the Message Queuing Interface or MQI, so that programs developed on one platform can readily be transferred to another.

This book describes the system administration aspects of MQSeries for IRIX and the services it provides to support commercial messaging in an SGI IRIX environment. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Who this book is for

Primarily, this book is for system administrators, and system programmers who manage the configuration and administration tasks for 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 SGI IRIX operating system, and utilities associated with it. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

How to use this book

Read [Chapter 1. "Introduction"](#) first for an understanding of MQSeries for IRIX.

The sections of this book contain information about:

- How to manage your MQSeries system:
    - Install and set up the product
    - Manage queue managers and queues
    - Monitor queue managers using instrumentation events
    - Set up security
    - Recover from a system failure
    - Analyze any problems that arise
- The MQSeries control commands, including railroad syntax diagrams.
- Sample resource definitions.

MQSeries for IRIX system restrictions

The following restrictions apply to the use of MQSeries for IRIX facilities with the MQSeries product:

1. MQCONN sets up its own signal handler for the signals:

        SIGSEGV
        SIGBUS

    User's handlers for these are restored after every MQI call.

    The remaining signals are handled differently.

        SIGINT
        SIGQUIT
        SIGFPE
        SIGTERM
        SIGHUP

    If any handler for this second group of signals receives an interrupt within an MQI call, the application handler must exit the application. MQI may not be called.

2. For each MQI call, MQSeries uses the UNIX interval timer ITIMER_REAL to generate SIGALRM signals. Any previous SIGALRM handler and timer interval is saved on entry to MQI and restored on exit. Any timer interval set is therefore frozen while within MQI.

MQSeries publications

Evaluating products

MQSeries Brochure, G511-1908

MQSeries: An Introduction to Messaging and Queuing, GC33-0805

MQSeries Message Queue Interface Technical Reference, SC33-0850

Planning

MQSeries Planning Guide, GC33-1349

MQSeries for MVS/ESA Version 1 Release 1.4 Licensed Program Specifications, GC33-1350

MQSeries for OS/400 Version 3 Release 1 (and later) Licensed Program Specifications, GC33-1360 (softcopy only)

Administration

MQSeries Clients, GC33-1632

MQSeries Command Reference, SC33-1369

MQSeries Programmable System Management, SC33-1482

MQSeries for AIX Version 2.2.1 System Management Guide, SC33-1373

MQSeries for HP-UX Version 2.2.1 System Management Guide, GC33-1633

MQSeries for SCO OpenServer and MQSeries for UnixWare Version 2 System Management Guide (Willow Technology publication)

MQSeries for IRIX Version 2 System Management Guide (Willow Technology publication)

MQSeries for OS/2 Version 2.0.1 System Management Guide, SC33-1371

MQSeries for SINIX and DC/OSx Version 2.2 System Management Guide, GC33-1768

MQSeries for SunOS Version 2.2 System Management Guide, GC33-1772

MQSeries for Sun Solaris Version 2.2 System Management Guide, GC33-1800

MQSeries for Windows NT Version 2.0 System Management Guide, SC33-1643

MQSeries for MVS/ESA Version 1 Release 1.4 Program Directory

MQSeries for MVS/ESA Version 1 Release 1.4 System Management Guide, SC33-0806

MQSeries for OS/400 Version 3 Release 2 Administration Guide, GC33-1361

MQSeries for OS/400 Version 3 Release 6 Administration Guide, SC33-1361

MQSeries for OS/400 Version 3 Release 6 Programmable Command Formats, SC33-1228

MQSeries Three Tier Administration Guide, SC33-1451

MQSeries Three Tier Reference Summary, SX33-6098

Application programming

MQSeries Application Programming Guide, SC33-0807

MQSeries Application Programming Reference, SC33-1673

MQSeries Application Programming Reference Summary, SX33-6095

MQSeries for OS/400 Version 3 Release 1 (and later) Application Programming Reference (RPG), SC33-1362

MQSeries for OS/400 Version 3 Release 6 Application Programming Reference (C and COBOL), SC33-1363

MQSeries Three Tier Application Design, SC33-1636

MQSeries Three Tier Application Programming, SC33-1452

MQSeries Three Tier Reference Summary, SX33-6098

Problem determination

MQSeries for AIX Version 2.2.1 System Management Guide, SC33-1373

MQSeries for HP-UX Version 2.2.1 System Management Guide, GC33-1633

MQSeries for IRIX Version 2 System Management Guide
(Willow Technology publication)

MQSeries for OS/2 Version 2.0.1 System Management Guide, SC33-1371

MQSeries for SCO OpenServer and MQSeries for UnixWare Version 2 System Management Guide (Willow Technology publication)

MQSeries for SINIX and DC/OSx Version 2.2 System Management Guide, GC33-1768

MQSeries for SunOS Version 2.2 System Management Guide, GC33-1772

MQSeries for Sun Solaris Version 2.2 System Management Guide, GC33-1800

MQSeries for Windows NT Version 2.0 System Management Guide, SC33-1643

MQSeries for MVS/ESA Version 1 Release 1.4 Messages and Codes, GC33-0819

MQSeries for MVS/ESA Version 1 Release 1.4 Problem Determination Guide, GC33-0808

MQSeries for OS/400 Version 3 Release 2 Administration Guide, GC33-1361

MQSeries for OS/400 Version 3 Release 6 Administration Guide, SC33-1361

MQSeries Three Tier Administration Guide, SC33-1451

Special topics

MQSeries Distributed Queuing Guide, SC33-1139

Information about MQSeries on the Internet

The Willow Technology home page
The URL of the Willow Technology home page is: http://www.willowtech.com

The IBM MQSeries home page
The URL of the MQSeries product family home page is: http://www.software.ibm.com/ts/mqseries/

Part 1. Guidance section

Chapter 1. Introduction

This chapter introduces MQSeries for IRIX 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"

- "Objects"

- "System default objects"

- "Administration"

- "Clients and servers"

- "Extending queue manager facilities"

- "Security"

MQSeries and message queuing

MQSeries for IRIX applications use message queuing to participate in message-driven processing. Applications can communicate across different platforms by using the appropriate message queuing software products. For example, IRIX and MVS/ESA applications can communicate through MQSeries for IRIX and MQSeries for MVS/ESA respectively. The applications are shielded from the mechanics of the underlying communications.

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

The MQI is described in detail in the *MQSeries Application Programming Reference* manual.

Time-independent applications

With message queuing, the exchange of messages between the sending and receiving programs is time independent. This means that the sending and receiving applications are decoupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of

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

Message-driven processing

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

Messages and queues

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

What messages are

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

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

The format of the message descriptor is defined by MQSeries for IRIX. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference* manual.

Message lengths

In MQSeries for IRIX, the maximum message length is 4MB (where 1MB 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 queues are

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

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

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

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

Predefined and dynamic queues

Queues can be characterized by the way they are created:

- *Predefined queues* are created by an administrator using the appropriate command set. For example, the MQSC command DEFINE QLOCAL creates a predefined local queue. Predefined queues are permanent; they exist independently of the applications that use them and survive MQSeries for IRIX restarts.

- *Dynamic queues* are created when an application issues an open request specifying the name of a model queue. The queue created is based on a template queue definition, which is the model queue. You can create a model queue using the MQSC command DEFINE QMODEL. The attributes of a model queue, for example the maximum number of messages that can be stored on it, are inherited by any dynamic queue that is created from it.

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

Retrieving messages from queues

In MQSeries for IRIX, suitably authorized applications can retrieve messages from a queue according to these 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 **MQOPEN** request from the application determines the method used.

Objects

Many of the tasks described in this book involve manipulating MQSeries *objects.* In MQSeries for IRIX, there are four different types of objects:

- Queue managers, see "MQSeries queue managers".

- Queues, see "MQSeries queues".

- Process definitions, see "Process definitions".

- Channels, see ["Channels"](#).

Object names

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

For the other types of objects, each object has a name associated with it and can be referenced in MQSeries for IRIX 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*, that have a maximum of 20 characters. For more information about names see ["Names"](#).

Managing objects

MQSeries provides commands for creating, altering, displaying, and deleting objects. These include:

- MQSeries commands (MQSC), which can be typed in from a keyboard or read from a file.

- Programmable Command Format (PCF) commands, which can be used in a program.

For more information, see [Chapter 4. "Understanding administration command sets"](#).

Object attributes

The properties of an object are defined by its attributes, some of which 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 two ways of referring to an attribute:

- 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 MQSC facility is an important part of this book, you are more likely to see the MQSC 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 simply a queue that belongs to another queue manager. A *remote queue manager* is any queue manager, other than the local queue manager. A remote queue manager may exist on a remote machine across the network or it may exist on the same machine as the local queue manager. MQSeries for IRIX supports multiple queue managers on the same machine.

MQI calls

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

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

MQSeries queues

Queues are defined to MQSeries using the appropriate MQSC DEFINE command or the PCF Create Queue command. The command specifies the type of queue and its attributes. For example, a local queue object has attributes that specify what happens when applications reference that queue in MQI calls.   Examples of attributes are:

- Whether applications can retrieve messages from the queue (GET enabled).

- Whether applications can put messages on the queue (PUT enabled).

- Whether access to the queue is exclusive to one application or shared between applications.

- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth).

- The maximum length of messages that can be put on the queue.

For further details about defining queue objects, see the *MQSeries Command Reference* or the *MQSeries Programmable System Management* manual.

Using queue objects

In MQSeries, there are four types of queue object. Each type of object can be manipulated by MQSeries for IRIX commands and is associated with real queues in different ways:

1. A *local queue* object identifies a local queue belonging to the queue manager to which the application is connected. All queues are local queues in the sense that each queue belongs to a queue manager and, for that queue manager, the queue is a local queue.

2. A *remote queue* object identifies a queue belonging to another queue manager. This queue must be

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

You must also define a transmission queue and channels between the queue managers, before applications can send messages to a queue on another queue manager.

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

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

4. A *model queue* object 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 queues used by MQSeries

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

Application queues

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

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

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"](#), and [runmqtrm (Start trigger monitor)](#). For more information about triggering, see the *MQSeries Application Programming Guide.*

Transmission queues

A *transmission queue* temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration; see ["Remote administration"](#). For information about the use of transmission queues in distributed queuing, see the *MQSeries Distributed Queuing Guide.*

Dead-letter queues

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

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

Command queues

The command queue, named SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSeries for IRIX 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.

You can define a command queue for each queue manager by running the supplied command file `amqscoma.tst`.

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

MQSeries for IRIX supports instrumentation events, which can be used to monitor queue managers independently of MQI applications. Examples of conditions generating instrumentation events include:

- An application attempts to put a message on a queue that is not available or does not exist.

- A queue becomes full.

- A channel is 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 for IRIX queue manager. See "Initiation queues" for more information.

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

Use 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.

For information on channels and how to use them, see the *MQSeries Distributed Queuing Guide*, and also "Preparing channels and transmission queues for remote administration".

System default objects

The *system default objects* are a set of object definitions that can be created for each queue manager, using the command file `amqscoma.tst`, which is supplied with MQSeries. 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; 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, the attributes you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

Administration

In MQSeries, you carry out administration tasks by issuing *commands*. Three command sets are provided, depending on which tasks you want to perform and how you want to perform them. The command sets are described in [Chapter 4. "Understanding administration command sets"](). Administration tasks include:

- 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 Distributed Queuing Guide*.

Local and remote administration

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

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

Clients and servers

MQSeries for IRIX supports client-server configurations for MQI applications.

An *MQI 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.

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 MQI applications as well.

The difference between an MQI server and an ordinary queue manager is that a server has a dedicated communications link with each client. For more information about creating channels for

clients and servers, see the *MQSeries Distributed Queuing Guide*

You can run MQI clients from all client systems.

MQI applications in a client-server environment

When linked to a server, client MQI applications can issue MQI calls in the same way as local applications. The client application issues an **MQCONN** call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager. You must link your applications to the appropriate client libraries. See the *MQSeries Application Programming Guide* for further information.

Extending queue manager facilities

The facilities provided by a queue manager can be extended by:

- User exits

- Installable services

User exits

User exits provide a mechanism for users to insert their own code into a queue manager function. Two types of user exits are supported:

1. *Channel exits,* which change the way that channels operate.

2. *Data conversion exits,* which create source code fragments that can be put into application programs to convert data from one format to another.

Both types of exit are related to distributed queueing. For more information about these exits and how to use them, see the *MQSeries Distributed Queuing Guide.*

Installable services

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

An implementation of an installable service is called a *service component.* You can use the components supplied with the product, or you can write your own component to perform the functions that you require. Currently, the following installable services are provided:

- The **authorization service**, which allows you to build your own security facility. The default service component that implements the service is the Object Authority Manager (OAM), which is supplied with the product. By default, the OAM is active, that is, you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM.

- The **name service**, which allows queue managers to share queues.

See the *MQSeries Programmable System Management* manual.

Security

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

Chapter 2. Installing MQSeries for IRIX

This chapter tells you how to install MQSeries for IRIX and how to verify that your installation has been successful.

Note
Throughout this chapter and the remainder of the book, the name `mqmtop` has been used to represent the name of the installation directory. For MQSeries for IRIX, the name of the actual directory is **/opt/mqm**

Components you can install

When you install MQSeries for IRIX you can choose which components to install. The components are as follows:

Component	Purpose
Server	Support for servers. Requires the base product to be installed.
Man	Man pages for control commands, MQI calls, and MQSC commands.
IRIX client	Support for IRIX clients. To install a client, the code must be copied to the client machine or machines.
Desktop client	Support for OS/2, DOS, and Windows 3.1 clients with an MQSeries for IRIX server. To install a client, the relevant code must be copied to the client machine or machines.
Samples	Samples
French	MQSeries messages - French
German	MQSeries messages - German
Spanish	MQSeries messages - Spanish

Typically a particular IRIX machine is designated as either an MQSeries client or server. This means that in most cases you install either the client component or the server component.

Note: The "base" product is automatically installed.

Installation requirements

The installation requirements depend on which components you install and how much working space you need. This, in turn, depends on the number of queues that you use, the number and size of the messages on the queues, and whether the messages are persistent or not. You also require archiving capacity on disk, tape, or other media.

Disk storage

These are the approximate storage requirements:

Server	A minimum of 10MB of disk space must be available for the product code and data in the filesystem containing the `/opt/mqm` directory.
	An additional 20MB of disk space must be available for MQSeries objects and data on the `/var/mqm` filesystem.
Clients	If you are installing client code, the storage required on the client machines is:
	IRIX client
	1.5MB on the IRIX machine.
	DOS
	240KB on the DOS machin
	OS/2
	1.2MB on the OS/2 machineWindows 3.1552KB on the Windows machine

Working data for MQSeries for IRIX is stored by default in `/var/mqm`. See <u>Figure 34</u> for the directory structure.

Note: For added confidence in the integrity of your data, you are strongly advised to put your logs onto a *different* physical drive from the one that you use for the queues.

Preparing for installation

This section guides you through some of the steps you must perform before you install MQSeries for IRIX.

Before installation...

Before you can install MQSeries for IRIX you:
 	
  • Must create a group with the name `mqm`
 	
  • Must create a user ID with the name `mqm`.
 	
  • Are recommended to create and mount a `/var/mqm` filesystem, or `/var/mqm`, `/var/mqm/logs`, and `/var/mqm/errors` filesystems.
 	
      The filesystems for MQSeries data **must** be on a local filesystem that supports long names, for example, `efs` or `xfs`.

After installation, this user ID (`mqm`) owns the directories and files that contain the resources associated with the product. This group and user must be defined for any machine on which the MQSeries software is to be installed, whether the machine is a client or a server machine.

If you want to run any administration commands, for example, **crtmqm** (create queue manager) or **strmqm** (start queue manager), your user ID must be a member of group `mqm`.

For stand-alone machines, you can create the new user and group IDs locally. For machines administered in a network information services (NIS) domain, you can create the user and group IDs on the NIS master server machine.

Installation

The MQSeries product is contained in the root directory (/) of the CD-ROM, and the product documentation (postscript and Adobe Acrobat) files are contained in the `/docs` subdirectory.

Notes: 1.   The manuals take up considerable disk space and are therefore not installed by the installation program. The user should copy required documentation files as needed.

          2.   The filenames are listed in "Directories that exist after installation"

To install, carry out the following procedure:

Installing Software From a Local CD-ROM Drive

1. Insert the CD in the CD-ROM drive.

2. Double-click the CD drive icon to open the Software Manager.

3. To read the release notes, see "Viewing Release Notes From a Local CD" later in this section. *Before installing software, it is recommended that you review the release notes because some patches that are not automatically installed are required.*

4. To install all products, mark both Upgrade Products and New Products for installation; otherwise press *Customize Installation* and mark the desired products.

5. Click the *Start* button.

6. When the installation is complete, choose "Quit" from the File menu.

7. Eject the CD from the drive
    - With the CD drive icon selected, hold down the right mouse button and select "Eject CDROM" from the desktop menu.

For more information about software installation, including how to customize your installation, see the online *Personal System Administration Guide.* This guide is located in the *IRIS Insight Library*, which you can find by selecting Online Books from the Help toolchest.

Installing Software From a Remote CD-ROM Drive

1. Insert the CD in the remote CD-ROM drive.

2. Select Software Manager from the System toolchest

3. In the Available Software field, type the remote system's hostname, a colon(:), and "/CDROM/dist". For example, type:

4. devhost:/CDROM/dist

5. Click the *Lookup* button.

6. To read the release notes, see "Viewing Release Notes From a Remote CD" later in this section. *Before installing software, it is recommended that you review the release notes because some patches that are not automatically installed are required.*

7. To install all products, mark both Upgrade Products and New Products for installation; otherwise press *Customize Installation* and mark the desired products.

8. Click the *Start* button.

9. When the installation is complete, choose "Quit" from the File menu.

10. Eject the CD from the drive

- With the CD drive icon selected, hold down the right mouse button and select "Eject CDROM" from the desktop menu.

For more information about software installation, including how to customize your installation, see the online *Personal System Administration Guide*. This guide is located in the *IRIS Insight Library*, which you can find by selecting Online Books from the Help toolchest.

Installing Software on a Non-Graphics System

If you are using an ASCII terminal or shell window, you can use the *inst*(1M) command to install the software from a local or remote CD. See *IRIX Admin: Software Installation and Licensing*.

Viewing Release Notes from a Local CD
(with Indigo magic Desktop)

After the CD is inserted in the CD drive, follow these steps to access the release notes.

1. Software Manager should be displayed and the "Available Software" field set to **"/CDROM/dist"**/

2. Click the *Customize Installation* button.

3. After the Software Inventory is displayed, highlight the product name whose release notes you wish to read by clicking only once on the product name.

4. Select "Release Notes" from the Selected menu. This displays the release notes in a window labeled grelnotes.

5. To print the release notes, select "Print" form the grelnotes File menu.

6. To read the release notes of another product in the Software Inventory, exit the release notes of the currently displayed product and repeat steps 3 and 4.

Viewing Release Notes from a Remote
(with Indigo Magic Desktop)

After the CD is inserted into a remote CD drive, follow these steps to access the release notes.

1. Software Manager should be displayed and the "Available Software" field set to the remote system's hostname, a colon(:), and **"/CDROM/dist"**/. For example, type:

devhost:/CDROM/dist

2. Click the *Customize Installation* button.

3. After the Software Inventory is displayed, highlight the product name whose release notes you wish to read by clicking only once on the product name.

4. Select "Release Notes" from the Selected menu. This displays the release notes in a window labeled grelnotes.

5. To print the release notes, select "Print" form the grelnotes File menu.

6. To read the release notes of another product in the Software Inventory, exit the release notes of the currently displayed product and repeat steps 3 and 4.

1. `mount` the CD-ROM as the `/cdrom` directory.

2. use the `"/usr/sbin/swmgr"` program, and carry out the following procedure:

3. Perform a.

4. When you are prompted for which packages are to be installed, select the ones you require. If you want to install the entire MQSeries product, select `all`.

5. Press the `Enter` key.

      1.

Notes: 1. If you have previously installed MQSeries on your system, you need to remove the product using the `pkgrm` program.

2. If the product is present, but not installed correctly, you may need to manually delete the files and directories contained in:
`/var/mqm`
`/opt/mqm`

Kernel configuration

MQSeries makes use of semaphores and shared memory and the **default** kernel configuration is *not adequate.*

In particular, the default number of semaphores and shared memory segments are *not* sufficient to support MQSeries.

If you attempt to use MQSeries without increasing SEMMNI, the number of semaphores, the queue manager fails and produces a First Failure Support Technology (FFST) file. This file indicates the error "'No space left on device' from semget" in Comment1.

If the value of shmmax is too low, the error "'Invalid argument' from shmget" will be received.

If the value of sshmseg is too low, the error "'Too many open files' from shmat" will be received..

For more information on viewing FFST files see ["How to examine the FFSTs"](#).

The following are the minimum recommended additional amounts for MQSeries (add the values to those you already have defined):

IRIX 6.2, 6.3 and 6.4:

```
semmni      240
semmns      300
semmnu      200
semmsl      256
semopm      256
semume      256
shmmnu      512
sshmseg     512
```

IRIX 6.5:

```
semmni      240
semmsl      256
semopm      256
shmmnu      512
sshmseg     512
```

Once you have made the changes, you must rebuild the kernel and restart the system.

Note: A heavily loaded/configured Queue Manager many require resources in excess of those recommneded above.

Before reconfiguring the ketnel, make a backup of your unix kernel (i.e. cp /unix /unix.old)

To change the kernel tunables, use systune in interactive mode (i.e. systune –i). Systune will isse a prompt (systune->). Enter the parameter name you wish to change followed by a space and the value you wish to assign to the parameter (e.g. systune->semmni 300) . Press enter. A 'y' response is required in response to the message:

"Do you really want to change <parameter> to <value> (<hex value>)? (y/n)

When you have completed you modifications to the tunables, enter "/etc/autoconfig –f" to rebuild the kernel with the new values and reboot.

Directories that exist after installation

When you install MQSeries for IRIX, the following directories are created or added to the product files:

For information about the other MQSeries directories see Figure 34.

When you install the product, you also receive:

- A set of man pages for the following entities:

    - MQSeries control commands

    - MQSeries commands (MQSC)

    - Message Queue Interface (MQI) calls.

- A set of MQSeries books is provided in PostScript and Adobe Acrobat formats. These enable you to print the books on a PostScript printer, or view the book with a suitable viewer. The filenames of the files and the names of the corresponding books are:

Filename	Book
AppPrgGde	MQSeries Application Programming Guide
AppPrgRef	MQSeries Application Programming Reference
Clients	MQSeries Clients
CmdRef	MQSeries Command Reference
DistQGde	MQSeries Distributed Queuing Guide
PrgSysMgmt	MQSeries Programmable System Management
IRIX_SMG	MQSeries for IRIX Version 2 System Management Guide

Translated messages

Messages in US English are always available. If you require another of the languages that is supported by MQSeries for IRIX Version 2, you *must* ensure that your NLSPATH environment variable includes the appropriate directory.

For example:

```
export LANG=german
export NLSPATH=/usr/lib/locale/%L/LC_MESSAGES/%N
```

Figure 3. Default directory structure for the product files

```
/mqmtop ──┬── bin/
          │
          ├── inc/
          │
          ├── lib/
          │
          ├── samp/ ──┬── bin/
          │           │
          │           └── dlq/
          │
          ├── os2_client/ ──┬── bin/
          │                 │
          │                 ├── dll/
          │                 │
          │                 ├── inc/
          │                 │
          │                 ├── lib/
          │                 │
          │                 ├── msg/
          │                 │
          │                 └── samp/
          │
          ├── dos_client/ ──┬── bin/
          │                 │
          │                 ├── inc/
          │                 │
          │                 ├── lib/
          │                 │
          │                 ├── msg/
          │                 │
          │                 └── samp/
          │
          └── win_client/ ──┬── bin/          ┌── man/ ──┬── man 1
                            │                 │          │
                            ├── dll/          │          └── man 3
                            │                 │
                            ├── inc/          │
                            │                 │
                            ├── lib/          │
                            │                 │
                            ├── msg/          │
                            │                 │
                            └── samp/ ────────┘
```
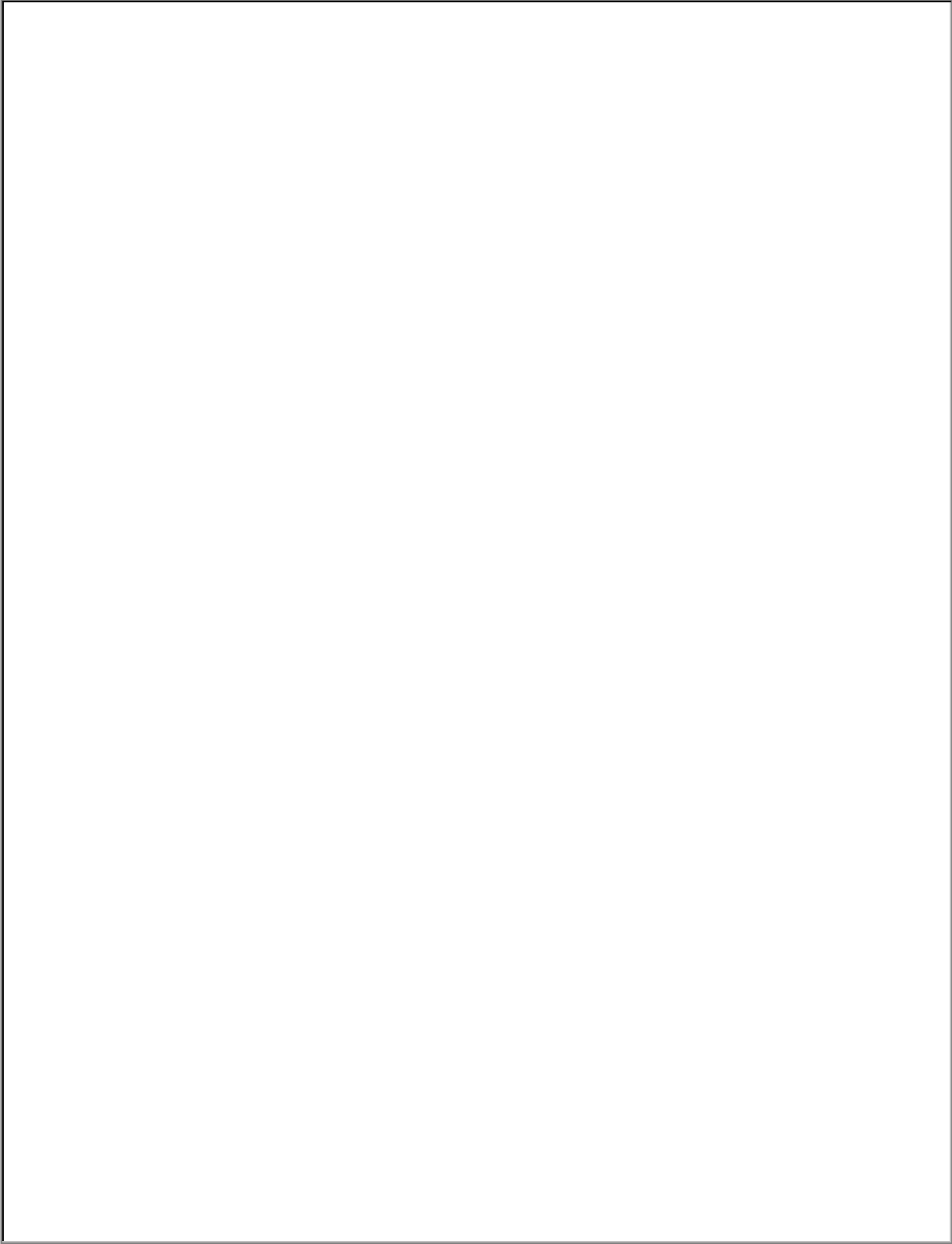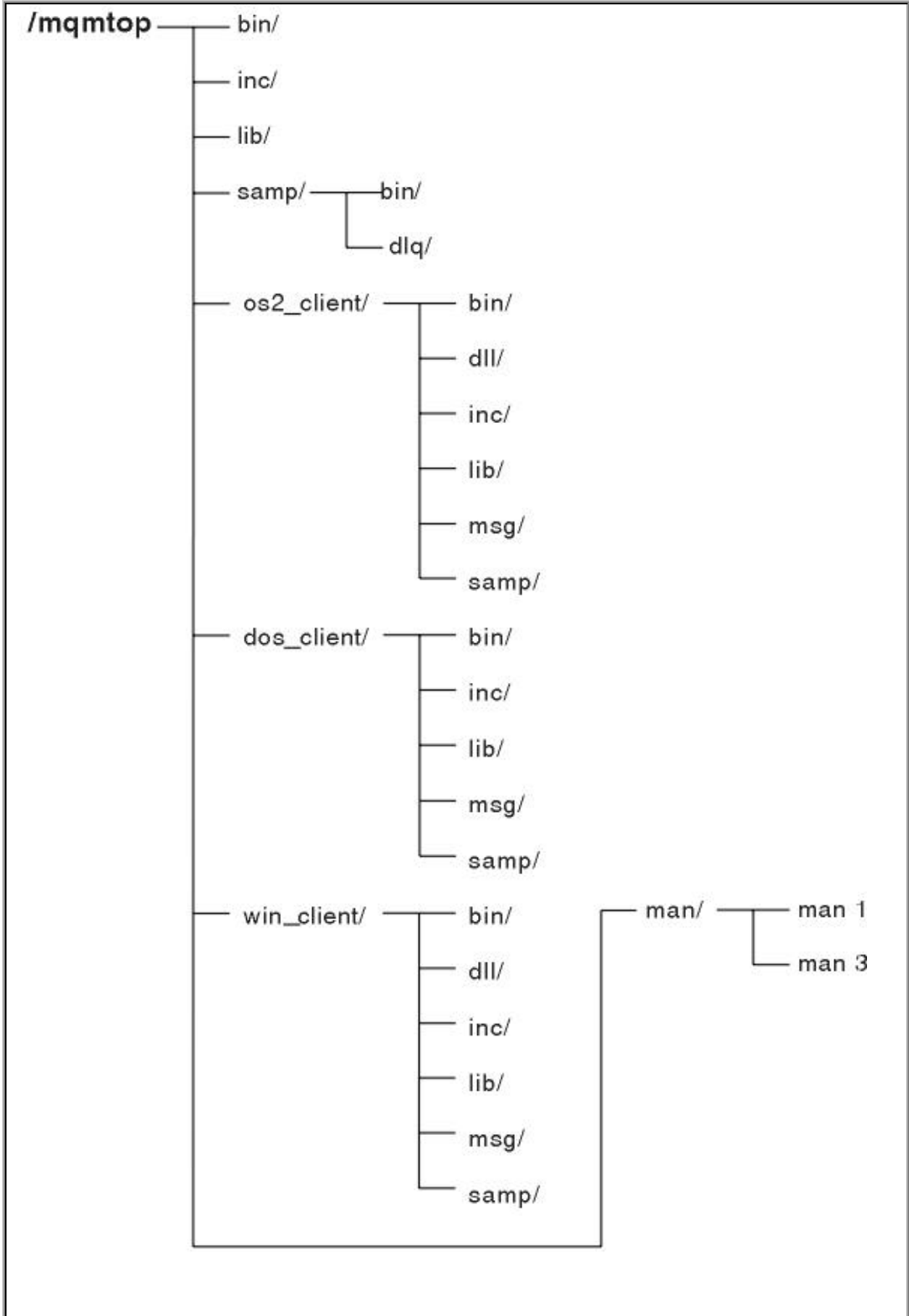
Verifying your installation

When you have installed the MQSeries for IRIX base, server and samples components, you should verify that the installation has completed successfully. The steps below tell you how to do this using the MQSC command file `amqscoma.tst`. The commands in this file initialize your MQSeries system and set up the default objects that your system requires. The objects that `amqscoma.tst` creates for you are listed in [Appendix B. "System defaults"](#).

When you have completed the verification, you should delete the queue manager to leave a 'clean' system, that is, a system with no objects, including queue managers, defined.

Note: Deleting the queue manager does not delete the installation. You can, therefore, use this procedure even if it has been run before, by you or someone else.

Follow these steps to verify your installation

The commands shown here are case-sensitive; type each command exactly as you see it.

These instructions assume that you are creating a queue manager called `QMNAME`. If you are creating a different queue manager, replace each occurrence of `QMNAME` with the chosen name in the following steps. Remember that a queue manager name must be unique within your network.

1. Create a queue manager called QMNAME using this command:

```
crtmqm -q QMNAME
```

   Notes:

   1. The queue manager name is case-sensitive.

   2. The -q flag denotes that this is the default queue manager. It is not essential to have a default queue manager, but it is recommended practice to have one.

   3. For a detailed description of the **crtmqm** command and options see [crtmqm (Create queue manager)](#).

2. Start the queue manager using this command:

```
strmqm QMNAME
```

   The **strmqm** command returns control when the queue manager has started and is ready to accept connect requests.

3. Create the default objects for this queue manager by typing this command:

```
runmqsc QMNAME < mqmtop/samp/amqscoma.tst > defobj.out
```

The file `amqscoma.tst` contains a series of MQSC commands that define the system default objects for the queue manager QMNAME. The output from the MQSC commands is sent to a report file `defobj.out`. Examine the last two lines of the output file to verify that all commands were processed without error. If errors have occurred, you should examine the rest of this file, checking the confirmation messages for each MQSC command. For example:

```
AMQ8006 MQSeries queue created
```

If no errors are indicated, all commands were successful and you have verified that your installation was successful.

You may wish to modify a copy of `amqscoma.tst` to meet your own requirements for system defaults. If you do, you may have to reinstall the toolkit component of the product later.

4. Stop the queue manager using the command:

```
endmqm QMNAME
```

5. Delete the queue manager using the command:

```
dltmqm QMNAME
```

This command deletes the queue manager and its associated objects including the system default objects that you created in step 3.

If your installation was not a success?

If any of the commands, including those run from the file `amqscoma.tst`, were not successful, look at the following:

- **Did you type in the commands correctly?** Try running one or more of the commands again. Remember that these commands and most parameters are case-sensitive. If you create a queue manager with an upper-case name, you must specify this as an upper-case name in any commands referring to this queue manager. For example, if you create a queue manager called QMNAME, you cannot use 'qmname' or 'QMname'.

- **Do you have enough disk space or memory to run the verification?** Check any error messages for an indication of this. If error message `AMQ7065 Insufficient space on`

disk is returned, use the **df -k** command to display the current size of the /var directory. If there is less than 15 MB of free space, increase the size of your /var file system.

- **Do the required directories for the installed product exist?** Check these, and if they do not exist, attempt to reinstall.

- **Do you have the required authority to run the commands?** Check that your user ID is a member of group mqm. If you are using /etc/logingroup, ensure that your ID is listed in there as belonging to the mqm group.

Installing clients

When you install MQSeries for IRIX, the files for the following clients are also provided. They are:

- OS/2

- DOS

- Windows 3.1

You can install MQSeries client software directly from CD-ROM, or from the LAN.

To install client software, install the component on an MQSeries for IRIX machine in the usual way and then copy the client files to the build and run environment on the client platform.

The required files are located in these directories:

- The DOS files are in these directories:
  ```
  mqmtop/dos_client/bin
  mqmtop/dos_client/inc
  mqmtop/dos_client/lib
  mqmtop/dos_client/msg
  mqmtop/dos_client/samp
  ```

- The OS/2 files are in these directories:
  ```
  mqmtop/os2_client/bin
  mqmtop/os2_client/dll
  mqmtop/os2_client/inc
  mqmtop/os2_client/lib
  mqmtop/os2_client/msg
  mqmtop/os2_client/samp
  ```

- The Windows 3.1 files are in these directories:
  ```
  mqmtop/win_client/bin
  mqmtop/win_client/dll
  mqmtop/win_client/inc
  mqmtop/win_client/lib
  mqmtop/win_client/msg
  mqmtop/win_client/samp
  ```

Further information about clients can be found in the *MQSeries Clients* book.

Chapter 3. Customizing your system

This chapter lists the tasks involved in customizing a queue manager to meet your requirements.

Do I need to customize?

> When you have installed the product, you can use it without having to customize it in any way. The default configuration provides all the facilities you need to build a working system that can participate in message queuing with other MQSeries systems.
>
> However, you must have set up the required IRIX user and group IDs. See <u>"Specifying user groups for security administration"</u> for more information.

When do I customize?

> Some customization tasks must be performed **before** you create a queue manager; others require you to stop and restart the queue manager. Check each task in turn, to see when you need to perform it.

What are configuration files?

> There are two types of configuration files. One contains information about the way your MQSeries system is set up or configured; this file is created when MQSeries is installed. The other contains information about the attributes of an individual queue manager. This file is generated when a queue manager is created.
>
> <u>"Things you can customize"</u> specifies which of these files to modify for each relevant configuration task. For more information about the files themselves, see <u>Chapter 13. "Understanding configuration files"</u>.

What do I do now?

> Check each item in the list shown in <u>"Things you can customize"</u> to see if any of the things listed apply to your enterprise. If none of the items apply, you can ignore this chapter.

Things you can customize

Read through the following list to determine if any of the following aspects apply to systems within your enterprise:

- Configuring an authorization service component,

- Enabling communications support, see topic <u>"Enabling communications support"</u>.

- Implementing data conversion, see topic <u>"Implementing data conversion"</u>.

- Defining the default and system objects, see topic <u>"Defining the default and system objects"</u>.

- Specifying a default prefix for queue manager objects, see topic <u>"Specifying prefixes for queue manager objects"</u>.

- Specifying log parameters, see topic <u>"Specifying logging parameters"</u>.

- Configuring a queue manager, see topic <u>"Configuring a queue manager"</u>.

- User groups for security administration, see topic <u>"Specifying user groups for security administration"</u>.

The terms in this list are explained in the following sections.

Configuring an authorization service component

- *This task is not required on your first pass through this book.*

- *By default, authority checking is turned on.*

The authorization service supports authority checking on commands and MQI calls for the user ID associated with the command or call. The names of the authorization service and the component that implements the service are specified in the queue manager configuration file (`qm.ini`).

By default, the active authorization service component is the Object Authority Manager (OAM), which is supplied with the product.

Changing the authorization service component

You can edit the configuration file for a specific queue manager to:

- Remove the OAM and therefore all security checking.

- Replace the OAM with a user-written authorization service component.

- Add a user-written authorization service component to augment the OAM.

These tasks are not required, unless you have specific security requirements that cannot be accommodated by the OAM.

For more information about the queue manager configuration file, see <u>"Queue manager configuration file"</u>. For information about writing your own authorization service component, refer to the *MQSeries Programmable System Management* manual.

Note: You can change the configuration file `qm.ini` after you have created and started the queue manager to which it relates. This has no effect until the queue manager is stopped and restarted. However, you should not create or change objects when the authorization service is off and then turn authorization back on again. If you do, you may compromise the security of your system.

Enabling communications support

- *This task is required before you can communicate with other queue managers.*

> • *You must stop and restart the queue manager to perform this task.*
>
> • *By default, the communications protocols are not defined.*

You must specify the name of the communications protocol and other parameters that are to be used for communication with other queue managers. This includes the LAN protocol name, which must be:

- • TCP/IP

Note: SNA LU6.2 is not supported on MQSeries for IRIX.

Implementing data conversion

> • *This task is not normally required on your first pass through this book.*
>
> • *You do not need data conversion to communicate between similar nodes.*

If you are using MQSeries with systems that have different encodings, you need to use a data conversion exit. The conversion of messages is based on message formats--specified in the message descriptor--and all message queuing formats are converted automatically. However, user formats are not converted so that even ASCII-to-EBCDIC conversion must be done using an exit (one per format).

You can use the supplied conversion exit utility if you wish to communicate with queue managers using MQI calls or remote commands, where the systems involved have formats outside those supported by MQSeries. The conversion exit utility allows you to create the required conversions as C source code. Refer to the *MQSeries Distributed Queuing Guide* for more information. You can leave this task until run time. However, if you do, you may not be able to communicate between the two different machines until then.

Supported codesets

MQSeries for IRIX supports most of the codesets used by the locales that are provided as standard on IRIX.

Details of the supported codesets are given in <u>Appendix E. "Support for different codesets on MQSeries for IRIX"</u>.

Adding information about coded character sets on IRIX

MQSeries stores information about the coded character sets that your operating system supports. If future versions of your operating system support additional coded character sets, you may need to update the information that MQSeries stores.

To update coded character set information edit the file `ccsid.tbl` in the IRIX directory `/var/mqm/conv/table`.

Example of ccsid.tbl

```
# Licensed Materials - Property of IBM
#
# (C) Copyright IBM Corp. 1994, 1995
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# This file is used to add newly supported CCSID values to the system.
# To be used the file should be placed in the /var/mqm/conv/table
# directory. CCSID and encoding values van be found in the CDRA
# documentation (SC09-1390). For mixed CCSIDs the "SBCS part" field
# should be set to the SBCS CCSID, otherwise this field should be 0.
# The codeset name is the name by which this CCSID is known on this
# operating system.
#
# The file can also be used to modify data stored for existing CCSIDs.
#
# Decimal     Hexadecimal        Decimal        Codeset
# CCSID       Encoding system    SBCS part      name

850          0x2100             0              ib850
437          0x2100             0              ib437
```

Defining the default and system objects

> • *This task is required, but is part of the standard administration procedures,*
> *see Chapter 5. "Managing queue managers".*

MQSeries for IRIX provides an MQSC command file that you can use to set up the default and system objects. Typically, when you define an object, you do not define all the possible attributes. The ones you do not specify are inherited from the corresponding default object. The supplied command file amqscoma.tst, when used with the **runmqsc** command, creates a set of default and system objects. See "Running the supplied MQSC command files" for information about running this sample.

If you change the attributes of the default object, any objects of the same type you create inherit the new values.

Do not attempt this if you are not familiar with the different commands and command sets provided on MQSeries for IRIX.

Modifying the amqscoma.tst command file

You should consider modifying the command file amqscoma.tst if, for example:

- You have a large number of objects to create that have similar values that are not the same as those of the supplied default amqscoma.tst values.

- You have some specific requirements or limitations on the size of certain resources.

To modify `amqscoma.tst`, make a backup copy, make the required changes, and then use the new version of the file to create the default objects. See also "Creating the default and system objects".

Specifying prefixes for queue manager objects

> • *This task is not normally required on your first pass through this book.*
>
> • *By default, the prefixes are already set.*
>
> • *You should not perform this task if you have existing MQSeries objects.*

The prefix for queue manager objects specifies the first part of the path to the files associated with a queue manager. You specify this when you install the product. Then, when you create any objects, this prefix is the first part of the path to the files associated with those objects. The prefix is specified in the *QueueManager* stanza in the MQSeries configuration file, `mqs.ini`.

If you change this prefix, all the objects are created at the locations specified by the new prefix. Unless you change it, the default prefix for queue manager objects is: `/var/mqm`.

Warning!

To modify the locations of queue manager objects, you must update the ***QueueManager*** stanza in ***mqs.ini*** file <u>before</u> you create any objects. Do not change this stanza if you have already created objects for this queue manager.

Specifying a default prefix

You can specify a default prefix, so that when you create a new queue manager its prefix is taken from the default. The default prefix is specified in the *DefaultPrefix* stanza in the `mqs.ini` file. Unless you have changed it, the default prefix is: `/var/mqm`.

Specifying logging parameters

> • *This task is not normally required on your first pass through this book.*
>
> • *By default, the logging parameters are adequate.*
>
> • *You must stop and restart the queue manager to perform this task.*

The logging parameters determine the type and size of the logs your system will use. These are specified in the configuration files `mqs.ini` and `qm.ini`, which are read when a queue manager is started. See "Log configuration stanzas" for more information.

Note: User ID `mqm` and group `mqm` must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the logs files are in the default locations supplied with the product.

Sharing queues using the name service

> • *This task is not normally required on your first pass through this book.*
>
> • *You must stop and restart the queue manager to perform this task.*

The name service is an installable service that enables an application to access a queue on another queue manager as if it were a local queue. For MQI requests, applications can then treat this queue like a local queue, without being aware of its exact location.

The service name and the component to be invoked for that service are specified by stanzas in the qm.ini configuration file. By default, this service is not active. For information about configuration file stanzas and writing your own name service, see *MQSeries Programmable System Management* manual.

Configuring a queue manager

> • *This task is required, but is part of the standard administration procedures, see* [*Chapter 5. "Managing queue managers"*](#).

When you create a queue manager, using the **crtmqm** command, you can specify certain properties for that queue manager. For example, you can specify the name of the dead-letter queue, and the default transmission queue.

Once you have created a queue manager, you may need to modify its properties. For more information, see ["Guidelines for creating queue managers"](#) and [Chapter 13. "Understanding configuration files"](#).

Specifying user groups for security administration

> • *You* must *perform this task before you install the queue manager.*

You must create the user and group ID mqm before you install the product. (You should have already done this.) See ["Preparing for installation"](#) for more information.

At this stage, you should also consider creating groups for user IDs for applications and administrators. You can, however, perform this task at any time.

Chapter 4. Understanding administration command sets

Read this chapter for an overview of the different methods that you can use to perform system administration tasks on MQSeries objects. This chapter also helps you to understand the different methods and when each should be used.

Administration tasks include creating, starting, altering, viewing, stopping, and deleting MQSeries objects, that is, queue managers, queues, processes, and channels. To perform these tasks, you must select the appropriate command from one of the supplied command sets.

MQSeries for IRIX provides three command sets for invoking administration tasks:

- Control commands

- MQSC commands

- PCF commands

This chapter describes the command sets that are available and provides a summary of the different commands in "Comparing command sets".

Control commands

Control commands fall into three categories:

- *Queue manager commands*, including commands for creating, starting, stopping, and deleting queue managers and command servers.

- *Channel commands*, including commands for starting and ending channels and channel initiators.

- *Utility commands*, including commands associated with:

    - Running MQSC commands.

    - Conversion exits.

    - Authority management.

    - Recording and recovering media images of queue manager resources.

    - Displaying and resolving transactions.

    - Trigger monitors.

– Displaying the file names of MQSeries objects.

Using control commands

You type in control commands in an IRIX shell window. Control commands are case-sensitive, including the command name, the flags, and any arguments. For example, in the command:

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

- The command name is crtmqm not CRTMQM

- The flag is -u not -U

- The dead-letter queue is SYSTEM.DEAD.LETTER.QUEUE

- The argument is specified as jupiter.queue.manager; this is different from JUPITER.queue.manager.

Therefore, take care to type the commands exactly as you see them in the examples.

Chapter 15. "MQSeries control commands" describes the syntax and purpose of each command.

MQSeries commands (MQSC)

You use the MQSeries (MQSC) commands to manage queue manager objects, including the queue manager itself, channels, queues, and process definitions. For example, there are commands to define, alter, display, and delete a specified queue.

When you display a queue, using the DISPLAY QUEUE command, you display the queue *attributes*. For example, the MAXMSGL attribute specifies the maximum length of a message that can be put on the queue. The command does not show you the messages on the queue.

MQSC commands are available on other platforms including OS/2, AS/400, and MVS/ESA.

These commands are summarized in "Comparing command sets". For detailed information about each MQSC command, see *MQSeries Command Reference*.

Running MQSC commands

You run MQSC commands by invoking the control command **runmqsc** from an IRIX shell prompt. You can run MQSC commands:

- Interactively by typing them at the keyboard. See "Using the MQSC facility interactively".

- As a sequence of commands from an ASCII text file. See "Running MQSC commands from text files"

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

- *Verification mode*, where the MQSC commands are verified on a local queue manager, they are not actually run.

- *Direct mode*, where the MQSC commands are run on a local queue manager.

- *Indirect mode*, where the MQSC commands are run on a remote queue manager.

For more information about using the MQSC facility and text files, see "Using the MQSC facility interactively". For more information about the **runmqsc** command, see runmqsc (Run MQSeries commands).

PCF commands

The purpose of the 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. In fact, PCF commands cover the same range of functions that are provided by the MQSC facility. You can therefore 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 specifies a *management request.*

   Message format specifies *administration commands.*

Application data

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

The PCF message type specifies *command.*

The command identifier specifies the command, for example, `Change Queue.`

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 written in upper-case, for example RQMNAME, although they are not case-sensitive. These names are limited to 8 characters so that some attributes, for example QDPHIEV, are not easy to guess the meaning of. Objects attributes in PCF are written in italics, and are not limited to 8 characters, and are therefore easier to read. The PCF equivalent of RQMNAME, is *RemoteQMgrName* and of QDPHIEV is *QDepthHighEvent.*

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.

Comparing command sets

The following tables compare the facilities available from the different administration command sets.

Note: Only MQSC commands that apply to IRIX are shown.

Figure 4. Commands for queue manager administration

PCF	MQSC	Control
Change Queue Manager	ALTER QMGR	-
(Create queue manager)[*]	-	crtmqm
(Delete queue manager)[*]	-	dltmqm
Inquire Queue Manager	DISPLAY QMGR	-
(Stop queue manager)[*]	-	endmqm
Ping Queue Manager	PING QMGR	-
(Start queue manager)[*]	-	strmqm
Note: [*] Not available as PCF commands.		

Figure 5. Commands for command server administration

Description	Control
Display command server	dspmqcsv
Start command server	strmqcsv
Stop command server	endmqcsv

Note: Functions in this group are available only as control commands. There are no equivalent MQSC or PCF commands in this group.

Figure 6. Commands for queue administration

PCF	MQSC
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE
Clear Queue	CLEAR QUEUE
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE
Inquire Queue	DISPLAY QUEUE
Inquire Queue Names	DISPLAY QUEUE

Note: There are no equivalent control commands in this group.

Figure 7. Commands for process administration

PCF	MQSC
Change Process	ALTER PROCESS
Copy Process	DEFINE PROCESS(x) LIKE(y)
Create Process	DEFINE PROCESS
Delete Process	DELETE PROCESS
Inquire Process	DISPLAY PROCESS
Inquire Process Names	DISPLAY PROCESS
Note: There are no equivalent control commands in this group.	

Figure 8. Commands for channel administration

PCF	MQSC	Control
Change Channel	ALTER CHANNEL	-
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	-
Create Channel	DEFINE CHANNEL	-
Delete Channel	DELETE CHANNEL	-
Inquire Channel	DISPLAY CHANNEL	-
Inquire Channel Names	DISPLAY CHANNEL	-
Ping Channel	PING CHANNEL	-
Reset Channel	RESET CHANNEL	-
Resolve Channel	RESOLVE CHANNEL	-
Start Channel	START CHANNEL	runmqchl
Start Channel Initiator	START CHINIT	runmqchi

Start Channel Listener	-	rumqmlsr.
Stop Channel	STOP CHANNEL	-

Figure 9. Other control commands

Description	Control
Create MQSeries conversion exit	crtmqcvx
Display authority	dspmqaut
Display files used by objects	dspmqfls
Display MQSeries transactions	dspmqtrn
Record media image	rcdmqimg
Recreate media object	rcrmqobj
Resolve MQSeries transactions	rsvmqtrn
Run MQSC commands	runmqsc
Run trigger monitor	runmqtrm
Run client trigger monitor	runmqtmc
Set or reset authority	setmqaut
Note: Functions in this group are available only as control commands. There are no direct PCF or MQSC equivalents.	

Chapter 5. Managing queue managers

This chapter describes how you can perform operations on queue managers and command servers. It contains these sections:

- ["Guidelines for creating queue managers"](#)

- ["Stopping a queue manager"](#)

- ["Deleting a queue manager"](#)

- ["Managing the command server for remote administration"](#)

Getting started

Before you can do anything with messages and queues, you must create at least one queue manager. Once the installation process is complete, you can use the MQSeries control commands to create a queue manager and start it. Then you can use MQSC commands to create the required default objects and system objects. Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. You must create these objects for each queue manager you create. The supplied command file amqscoma.tst, when used with the **runmqsc** command, creates a set of default and system objects. See ["Running the supplied MQSC command files"](#) for information about running this sample.

See [Chapter 4. "Understanding administration command sets"](#) for more information about commands that can be used with MQSeries for IRIX, and the different methods of invoking them.

Guidelines for creating queue managers

A queue manager manages the resources associated with it, in particular the queues that it owns. It provides queueing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete MQSeries objects. You create a queue manager using the **crtmqm** command. However, before you try this, especially in a production environment, work through this checklist:

- Specify a unique queue manager name.

- Limit the number of queue managers.

- Specify a default queue manager.

- Specify a dead-letter queue.

- Specify a default transmission queue.

- Specify the required logging parameters.

- Back up configuration files after creating a queue manager.

The terms in this list are explained in the sections that follow.

Specify a unique queue manager name

When you create a queue manager, you must ensure that no other queue manager has the same name, anywhere in your network. Queue manager names are not checked at create time, and non-unique names will prevent you from creating channels for distributed queuing.

One method of ensuring uniqueness is to prefix each queue manager name with its own (unique) node name. For example, if a node is called `accounts`, you could name your queue manager `accounts.saturn.queue.manager`, where `saturn` identifies a particular queue manager and `queue.manager` is an extension you can give to all queue managers. Alternatively, you can omit this, but note that `accounts.saturn` and `accounts.saturn.queue.manager` are *different* queue manager names.

If you are using MQSeries for communicating with other enterprises, you can also include your own enterprise as a prefix. We do not actually do this in the examples, because it makes them more difficult to follow.

Note: Queue manager names in control commands are case-sensitive. This means that you could create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. Such complications are best avoided.

Limit the number of queue managers

In MQSeries for IRIX, you can create as many queue managers as resources allow. However, because each queue manager requires its own resources, it is generally better to have one queue manager with 100 queues than ten queue managers with ten queues each. In production systems, many nodes will be run with a single queue manager, but larger server machines may run with multiple queue managers.

Specify the default queue manager

Each node should have a default queue manager, though it is possible to configure MQSeries on a node without one.

To create a default queue manager, specify the -q flag on the **crtmqm** command. For a detailed description of this command and its parameters, see [crtmqm (Create queue manager)](#).

What is a default queue manager?

The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an **mqconn** call. It is also the queue manager that processes MQSC commands when you invoke the **runmqsc** command without specifying a queue manager name.

How do you specify a default queue manager?

You include the -q flag. on the **crtmqm** command to specify that the queue manager you are creating is the default queue manager. Omit this flag if you do not want to create a default queue manager.

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

What happens if I make another queue manager the default?

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

This may be what you intend, but you should take this into account before you change the default.

Specify a dead-letter queue

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

Warning!
It is vitally important to have a dead-letter queue on each queue manager in your network. Failure to do so may mean that errors in application programs cause channels to be closed or that replies to administration commands are not received.

For example, if an application attempts to put a message on a queue on another queue manager, but the wrong queue name is given, the channel is stopped, and the message remains on the transmission queue. Other applications cannot then use this channel for their messages.

The channels are not affected if the queue managers have dead-letter queues. The undelivered message is simply put on the dead-letter queue at the receiving end, leaving the channel and its transmission queue available.

Therefore, when you create a queue manager you should use the -u flag to specify the name of the dead-letter queue. You can also use an MQSC command to alter the attributes of a queue manager and specify the dead-letter queue to be used. See "Altering queue manager attributes" for an example of an MQSC ALTER command.

A sample dead-letter queue definition is provided with the supplied sample amqscoma.tst. The queue is called SYSTEM.DEAD.LETTER.QUEUE. See "Creating the default and system objects" for information about running this sample. When you find messages on a dead-letter queue, you can use the dead-letter queue handler, supplied with MQSeries, to process these messages. See Chapter 11. "The MQSeries dead-letter queue handler" for further information about the dead-letter queue handler itself, and how to reduce the number of messages that might otherwise be placed on the dead-letter queue.

Specify a default transmission queue

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

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

Specify the required logging parameters

You can specify logging parameters on the **crtmqm** command, including the type of logging, and the path and size of the log files. In a development environment, the default logging parameters should be adequate. However, you can change the defaults if, for example:

- You have a low-end system configuration that cannot support large logs.

- You anticipate a large number of long messages being on your queues at the same time.

For more information about specifying logging parameters:

- On the **crtmqm** command, see crtmqm (Create queue manager).

- Using configuration files, see "Log configuration stanzas".

Back up configuration files after creating a queue manager

There are two configuration files to consider:

1. When you install the product, the MQSeries configuration file (mqs.ini) is created. It contains a list of queue managers, which is updated each time you create or delete a queue manager. There is one mqs.ini file per node.

2. When you create a new queue manager, a new queue manager configuration file (qm.ini) is automatically created. This contains configuration parameters for the queue manager.

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

For more information about configuration files, see Chapter 13. "Understanding configuration files".

Working with queue managers

MQSeries provides control commands for creating, starting, ending, and deleting queue managers. You can also display a queue manager's attributes using the MQSC command DISPLAY QMGR

and change them using ALTER QMGR. See <u>"Displaying queue manager attributes"</u> and <u>"Altering queue manager attributes"</u>.

Creating a default queue manager

The following command creates a default queue manager called `saturn.queue.manager` and specifies the names of both its default transmission queue and its dead-letter queue:

```
crtmqm -q -d MY.DEFAULT.XMIT.QUEUE -u SYSTEM.DEAD.LETTER.QUEUE saturn.queue.manager
```

where:

-q

> **This queue manager is the default queue manager.**

-d MY.DEFAULT.XMIT.QUEUE

> **Name of the default transmission queue.**

-u SYSTEM.DEAD.LETTER.QUEUE

> **Name of the dead-letter queue.**

saturn.queue.manager

> **Name of this queue manager. For** crtmqm**, this must be the last parameter in the command.**

Starting a queue manager

Although you have created a queue manager, it cannot process commands or MQI calls until it has been started. Start the queue manager by typing in this command:

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

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

Creating the default and system objects

You must create a set of default and system objects for each queue manager you create. To do this, use the **runmqsc** command specifying both the name of the queue manager and the name of the command file containing the commands. (You can specify `amqscoma.tst`, which is supplied as part of the product.) The following command creates the default and system objects:

```
runmqsc saturn.queue.manager < mqmtop/samp/amqscoma.tst > defobj.out
```

You can run this command immediately after the **strmqm** command has completed.

The file `defobj.out` is created, if it does not already exist. When the command has completed, `defobj.out` contains the output from the MQSC file. You should check that all the commands ran successfully before continuing.

For more information about running the MQSC facility (**runmqsc**), see <u>"Running MQSC commands from text files"</u>.

Looking at object files

Each MQSeries queue, queue manager, or process object is represented by a file. Because these object names are not necessarily valid file names, the queue manager converts the object name into a valid file name, where necessary. This is described in <u>"Understanding MQSeries file names"</u>.

To find out how to display the real file name of an object, see <u>dspmqfls (Display MQSeries files)</u>.

Stopping a queue manager

To stop a queue manager, use the **endmqm** command. For example, to stop a queue manager called `saturn.queue.manager` use this command:

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

By default, this command performs a *controlled* or *quiesced* shutdown of the specified queue manager. This may take a while to complete--a controlled shutdown waits until *all* connected applications have disconnected.

Optionally, this **endmqm** command can have a flag that specifies how the shutdown is to be carried out.

If you have problems

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

- Do not check MQI return codes properly.

- Do not request a notification of a quiesce.

- Terminate without disconnecting from the queue manager (by issuing an **MQDISC** call).

Immediate and preemptive queue manager shutdowns

If a shutdown of a queue manager is very slow, or you believe that the queue manager is not going to stop, you can break out of the **endmqm** command using Ctrl-C. You can then issue another **endmqm** command, but this time with a flag specifying either an immediate or a preemptive shutdown.

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

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

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

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

Do not use this method unless all other attempts to stop the queue manager using the **endmqm** command have failed. This method can have unpredictable consequences for connected applications. If this method still does not work, see "Stopping a queue manager manually" for an alternative.

For a detailed description of the **endmqm** command and its options, see endmqm (End queue manager).

Restarting a queue manager

To restart a queue manager, use the command:

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

Making an existing queue manager the default

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

You may need to edit this stanza:

- • To make an existing queue manager the default. To do this you have to change the queue manager name in this stanza to the name of the new default queue manager. You must do this manually, using a text editor.

- • If you do not have a default queue manager on the node, and you want to make an existing queue manager the default. To do this you must create the *DefaultQueueManager* stanza--with the required name--yourself.

- • If you accidentally make another queue manager the default and wish to revert to the original default queue manager. To do this, edit the *DefaultQueueManager* stanza in the MQSeries configuration file, replacing the name of unwanted default queue manager with that of the one you do want.

See Chapter 13. "Understanding configuration files" for information about configuration files.

When the stanza contains the required information, stop the queue manager and restart it.

Deleting a queue manager

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

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

Deleting a queue manager is a drastic step, because you also delete all the resources associated with it. This includes not only all queues and their messages, but also all object definitions.

For a description of the **dltmqm** command and its options, see dltmqm (Delete queue manager). You should ensure that only trusted administrators have the authority to use this command.

If the usual methods for deleting a queue manager do not work, see "Removing queue managers manually" for an alternative.

Managing the command server for remote administration

Each queue manager has 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. There are separate control commands for starting and stopping the command server.

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. In fact, these messages are queued in the local transmission queue that serves the remote queue manager. This situation should be avoided, if at all possible.

Starting the command server

To start the command server use this command:

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

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

Displaying the status of the command server

For remote administration, you must ensure that the command server on the target queue manager is running. If it is not running, no remote commands can 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 command is:

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

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

```
AMQ8027    MQSeries Command Server Status ..: Running
```

Stopping a command server

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

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

You can stop the command server in two different ways:

- For a controlled stop, use the **endmqcsv** command with the -c flag.. This is the default.

- For an immediate stop, use the **endmqcsv** command with the -i flag.

Chapter 6. Administering local MQSeries objects

This chapter describes how to administer local MQSeries objects to support applications programs that use the Message Queuing Interface (MQI). In this context, local administration means creating, displaying, changing, copying, and deleting MQSeries objects.

This chapter contains these sections:

- "Supporting application programs that use the MQI"

- "Issuing MQSC commands for administration"

- "Running MQSC commands from text files"

- "If you have problems with MQSC"

- "Working with local queues"

- "Working with alias queues"

- "Working with model queues"

- "Managing objects for triggering"

Supporting application programs that use the MQI

MQI application programs need certain objects before they can run successfully. For example, Figure 10 shows an application that removes messages from a queue, processes them, and then sends some results to another queue on the same queue manager.
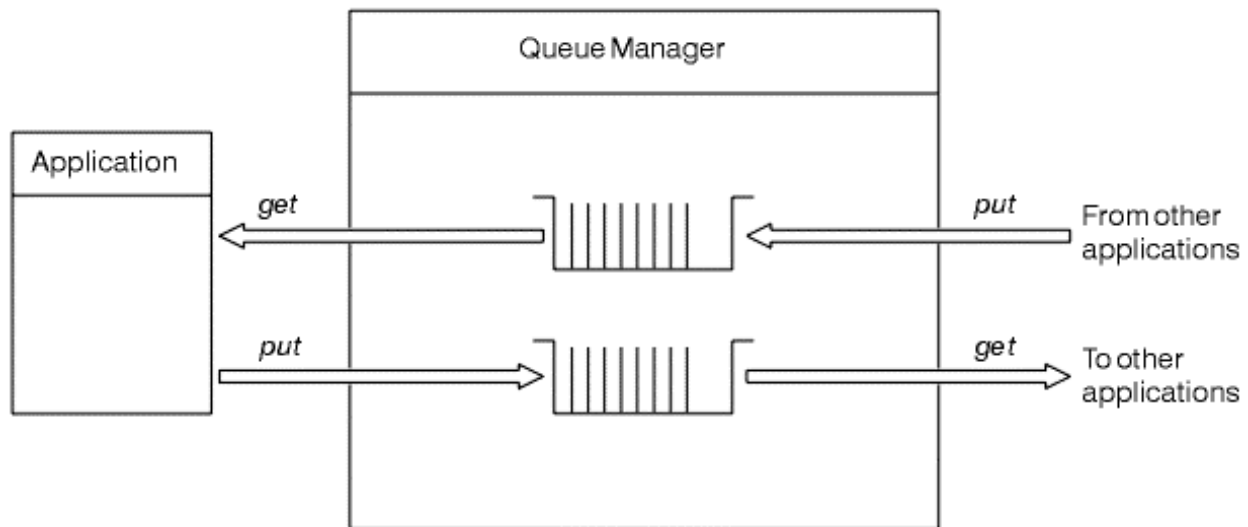
Whereas applications can put (using **MQPUT**) messages on local or remote queues, they can only get (using **MQGET**) messages directly from local queues.

Before this application can be run, these conditions must be satisfied:

- The queue manager must exist and be running.

- The first application queue, from which the messages are to be removed, must be defined.

- The second queue, on which the application puts the messages, must also be defined.

- The application must be able to connect to the queue manager. To do this it must be linked to the product code. See the *MQSeries Application Programming Guide* for more information.

- The applications that put the messages on the first queue must also connect to a queue manager. If they are remote, they must also be set up with transmission queues and channels. This part of the system is not shown in Figure 10.

Issuing MQSC commands for administration

In this section, we assume that you will be issuing commands using the **runmqsc** command. You can do this interactively--entering the commands at the keyboard--or you can redirect the standard input device (stdin) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same.

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

syntax.

Before you start

Before you can run MQSC commands, you must have created and started the queue manager that is going to run the commands, see <u>"Creating a default queue manager"</u>.

MQSeries object names

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

When you are issuing MQSC commands, you need only specify the local name of the queue. In our examples, we use queue names such as: ORANGE.LOCAL.QUEUE

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

We also use the name `saturn.queue.manager` as a queue manager name.

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

You do not have to use these names, but if you do not, you must modify any commands in examples that specify them.

Case-sensitivity in MQSC commands

MQSC commands, including their attributes, can be written in upper or lower case. Object names in MQSC commands are folded (that is, QUEUE and queue are not differentiated), unless the names are put in single quotes. If quotes are not used, the object is processed with a name in upper-case. See the *MQSeries Command Reference* manual for more information. However, the **runmqsc** command that invoke the MQSC facility is case-sensitive, see <u>"Using control commands"</u>.

Standard input and output

The *standard input device*, also referred to as `stdin`, is the device from which input to the system is taken. Typically, this is the keyboard, but you can redirect it to come from a serial port, a (disk) file, and so on. The *standard output device*, also referred to as `stdout`, is the device to which output from the system is sent. Typically, this is a display, but it can be redirected to a serial port, a file, and so on.

On a shell command the '<' operator redirects input. If this operator is followed by a file name, input is taken from the file. Similarly, the '>' operator redirects output; if this operator is followed by a file name, the output is sent to the file.

Using the MQSC facility interactively

To enter commands interactively, open a shell and type:

```
runmqsc
```

```

```

and then press Enter.

In this command, a queue manager name has not been specified, therefore, the MQSC commands will be processed by the default queue manager.   Now you can type in any MQSC commands, as required. For example, try this one:

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

Feedback from MQSC commands

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

```
AMQ8006: MQSeries queue created
 .
 .
 .
AMQ8405: Syntax error detected at or near end of command segment below:-
Z
```

The first message confirms that a queue has been created; the second indicates that you have made a syntax error. These messages are sent to the standard output device. If you have not entered the command correctly, refer to the *MQSeries Command Reference* manual for the correct syntax.

Ending interactive input to MQSC

If you are using MQSC interactively, you can exit by typing the EOF character CTRL+D.

If you are redirecting input from other sources, such as a text file, you do not have to do this.

Displaying queue manager attributes

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

```
DISPLAY QMGR ALL
```

A typical output is:

```
     1 : display qmgr all
AMQ8408: Display Queue Manager details.
   DESCR( )
   DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
   DEFXMITQ(MY.DEFAULT.XMIT.QUEUE)
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
   QMNAME(saturn.queue.manager)
   TRIGINT(999999999)
```

```
    MAXHANDS(256)
    MAXUMSGS(10000)
    AUTHOREV(DISABLED)
    INHIBTEV(DISABLED)
    LOCALEV(DISABLED)
    REMOTEEV(DISABLED)
    PERFMEV(DISABLED)
    STRSTPEV(ENABLED)
    MAXPRTY(9)
    CCSID(850)
    MAXMSGL(4194304)
    CMDLEVEL(100)
    PLATFORM(UNIX)
    SYNCPT
```

The ALL parameter on the DISPLAY QMGR command causes all the queue manager attributes to
be displayed. In particular, the output tells us the default queue manager name
(`saturn.queue.manager`), and the names of the dead-letter queue
(SYSTEM.DEAD.LETTER.QUEUE) and the command queue
(SYSTEM.ADMIN.COMMAND.QUEUE). Both these queues should have been created when you
ran the sample `amqscoma.tst`; see ["Creating the default and system objects"](#).

Before you go further, confirm that these queues have been created by typing the command:

```
DISPLAY QUEUE (SYSTEM.*)
or
DIS Q (SYSTEM.*)
```

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

Using a queue manager that is not the default

You can specify the queue manager name on the **runmqsc** command to run MQSC commands on a
local queue manager other than the default. For example, to run MQSC commands on queue
manager `jupiter.queue.manager` type:

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

and then press Enter.

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

You can also run MQSC commands on a remote queue manager, see ["Issuing MQSC commands
remotely"](#).

Altering queue manager attributes

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

example, use the following commands to alter the attributes of `jupiter.queue.manager`:

```
runmqsc jupiter.queue.manager

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

Press Enter after typing the first line.

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

Running MQSC commands from text files

Running MQSC commands interactively is suitable for quick tests, but if you have very long commands, or commands that you want to repeat, over again, you should redirect `stdin` from a text file. (See <u>"Standard input and output"</u> for information about `stdin` and `stdout`.) To do this, first create a text file containing the MQSC commands using your familiar text editor. When you use the **runmqsc** command, use the shell redirection operators. For example, the following command runs a sequence of commands contained in the text file `myprog.in`:

```
runmqsc < myprog.in
```

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

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

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

This command invokes the MQSC commands contained in the MQSC command file `myprog.in`. Because we have not specified a queue manager name, the MQSC commands are run against the default queue manager. The output is sent to the report file `myprog.out`. <u>Figure 11</u> shows an extract from the MQSC command file `myprog.in` and <u>Figure 12</u> shows the corresponding extract of the output in `myprog.out`.

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

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

and then press Enter.

MQSC command files

MQSC commands are written in human-readable form, that is, in ASCII text. <u>Figure 11</u> is an extract

from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. The *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

Figure 11. Extract from the MQSC command file, myprog.in

```
.
.
.
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
.
.
.
```

You must limit lines to maximum of 80 characters. The plus sign indicates that the command is continued on the next line.

MQSC reports

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

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

      Starting MQSeries Commands.

- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in . However, you can use the -e flag. on the **runmqsc** command to suppress the output.

- A syntax error message for any commands found to be in error.

- An *operator message* indicating the outcome of running each command. For example, the operator message for the successful completion of a DEFINE QLOCAL command is:

      AMQ8006: MQSeries queue created.

- Other messages resulting from general errors when running the script file.

- A brief statistical summary of the report indicating the number of commands read, the number of commands with syntax errors, and the number of commands that could not be processed.

Note: The queue manager will only attempt to process those commends that have no syntax errors.

Figure 12. Extract from the MQSC report file, myprog.out.

```
Starting MQSeries Commands.
 .
 .
    12:        DEFINE QLOCAL('RED.LOCAL.QUEUE') REPLACE   +
       :               DESCR(' ') +
       :               PUT(ENABLED) +
       :               DEFPRTY(0) +
       :               DEFPSIST(NO) +
       :               GET(ENABLED) +
       :               MAXDEPTH(5000) +
       :               MAXMSGL(1024) +
       :               DEFSOPT(SHARED) +
       :               USAGE(NORMAL) +
       :               NOTRIGGER
AMQ8006: MQSeries queue created.
       :
 .
 .
15 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Running the supplied MQSC command files

When you install MQSeries for IRIX, these MQSC command files are supplied:

amqscoma.tst

>   Default and system objects.

amqscos0.tst

>   Definitions of objects used by sample programs.

The files are located in the directory /**mqmtop**/samp.

You should already have run **runmqsc** against the command file amqscoma.tst. If you have not done this, or if you have deleted any of the objects created from it, run it again by typing:

```
runmqsc < /mqmtop/samp/amqscoma.tst
```

The DEFINE commands in amqscoma.tst specify the REPLACE option, which overwrites the existing definitions, if possible. See the *MQSeries Command Reference* manual for more information about REPLACE.

Using runmqsc to verify commands

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

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

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

- Running a large number of commands from a command file.

- Using an MQSC command file many times over.

This report is similar to that shown in .

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

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

the -w flag is ignored, and the command is run locally.

If you have problems with MQSC

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

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

- Check your file paths. By default, the **runmqsc** executable is located in directory /usr/bin, and this is linked to the product libraries using symlink.

- Use the indirection operator **<** when redirecting input from a file. Otherwise, the queue manager interprets the file name as a queue manager name. For example:

```
runmqsc amqscoma.tst

(C) Copyright Willow Technology, Inc. 1998. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

AMQ8118: MQSeries queue manager does not exist.
0 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

- If you redirect output to a file, use the **>** indirection operator. By default, the output goes to the directory from which you ran the **runmqsc** command. Specify a fully-qualified file name to send your output to a specific file and directory. For example:

```
runmqsc < /mqmtop/samp/amqscoma.tst > /u/zorg/myfile.output
```

- Check that you really have created the queue manager that is going to run the commands.

  To do this, look in the configuration file mqs.ini, which by default is located in the /var/mqm directory. This file contains the names of the queue managers and the name of the default queue manager, if you have one.

- The queue manager should already be started, if it is not, start it; see <u>"Starting a queue manager"</u>. You get an error message if it is already started.

- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, otherwise you get this error:

```
runmqsc <amqscoma.tst

(C) Copyright Willow Technology, Inc. 1998. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

AMQ8146: MQSeries queue manager not available.
0 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

  To correct this type of problem, see <u>"Making an existing queue manager the default"</u>.

- You cannot specify an MQSC command as a **runmqsc** parameter:

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

- You cannot enter MQSC commands from the shell before you issue the **runmqsc** command. For example:

```
DEFINE QLOCAL(QUEUE1)

ksh: DEFINE:     not found.
```

- You cannot run control commands from **runmqsc**. For example, you cannot start a queue manager once you are running MQSC interactively:

```
runmqsc
5697-176 (C) Copyright Willow Technology, Inc. 1998. ALL RIGHTS
RESERVED.
Starting MQSeries Commands.
```

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

See also ["If you have problems using MQSC remotely"](#).

Browsing queues

If you need to look at the contents of the messages on a queue, MQSeries for IRIX provides a sample queue browser for this purpose. The browser is supplied both as source and as a module that can be run. By default, the file names and paths are:

Source

> **mqmtop**/samp/amqsbcg0.c

Executable

> **mqmtop**/samp/bin/amqsbcg

The sample takes two parameters, the queue manager name (snooker) and the queue name (SYSTEM.ADMIN.RESPQ.tpp01). For example:

```
  amqsbcg snooker SYSTEM.ADMIN.RESPQ.tpp01
```

There are no defaults, both parameters are required. Typical results from this commands are:

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

 MQCONN to snooker
 MQOPEN - 'SYSTEM.ADMIN.RESPQ.tpp01'


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

  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'AMQMRESP'
  Priority : 5  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED47690071A6D00'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                             '
```

```
  ReplyToQMgr    : 'snooker                                          '
  ** Identity Context
  UserIdentifier : 'tpp01       '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                                '
  PutDate  : '19941124'    PutTime  : '11184015'
  ApplOriginData : '     '

****   Message       ****

 length - 268 bytes

00000000:   736E 6F6F 6B65 7220 2020 2020 2020 2020 'snooker         '
00000010:   2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000020:   2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000030:   534E 4F4F 4B45 522E 5749 4748 542E 5443 'SNOOKER.WIGHT.TC'
00000040:   5020 2020 2020 2020 2020 2020 2020 2020 'P               '
00000050:   2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:   0000 0001 0000 0024 0000 0001 0000 0015 '.......>........'
00000070:   0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000080:   0000 0003 0000 0004 0000 0028 0000 0DAD '...........(....'
00000090:   0000 0000 0000 0014 534E 4F4F 4B45 522E '........SNOOKER.'
000000A0:   5749 4748 542E 5443 5020 2020 0000 0003 'WIGHT.TCP   ....'
000000B0:   0000 0010 0000 05E7 0000 0001 0000 0004 '................'
000000C0:   0000 0050 0000 0DAE 0000 0000 0000 0039 '...P...........9'
000000D0:   2066 726F 6D20 736E 6F6F 6B65 7220 746F ' from snooker to'
000000E0:   2077 6967 6874 2076 6961 2074 6370 2F69 ' wight via tcp/i'
000000F0:   7020 2020 2020 2020 2020 2020 2020 2020 'p               '
00000100:   2020 2020 2020 2020 2000 0000          '         ...    '


 MQGET of message number 2
****Message descriptor****

  StrucId  : 'MD '  Version : 1
  Report   : 0  MsgType : 2
  Expiry   : -1  Feedback : 0
  Encoding : 273   CodedCharSetId : 850
  Format : 'MQADMIN '
  Priority : 8  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED476901524D200'
  CorrelId : X'414D5120736E6F6F6B657220202020202ED47690071A6D00'
  BackoutCount : 0
  ReplyToQ       : '                                          '
  ReplyToQMgr    : 'snooker                                          '
  ** Identity Context
  UserIdentifier : 'tpp01       '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
```

```
   PutApplName     : '                              '
   PutDate  : '19941124'     PutTime  : '11184035'
   ApplOriginData : '     '


****    Message       ****

 length - 36 bytes

00000000:  0000 0002 0000 0024 0000 0001 0000 0015  '.......>........'
00000010:  0000 0001 0000 0001 0000 0000 0000 0000  '................'
00000020:  0000 0000                                 '....            '


 MQGET of message number 3
****Message descriptor****

   StrucId  : 'MD '  Version : 1
   Report   : 0  MsgType : 8
   Expiry   : -1  Feedback : 0
   Encoding : 273   CodedCharSetId : 850
   Format : 'AMQMRESP'
   Priority : 5  Persistence : 1
   MsgId : X'414D5120736E6F6F6B657220202020202ED477D62A9EA100'
   CorrelId : X'000000000000000000000000000000000000000000000000'
   BackoutCount : 0
   ReplyToQ      : '                                '
   ReplyToQMgr   : 'snooker                         '
   ** Identity Context
   UserIdentifier : 'trevor      '
   AccountingToken :
    X'0437303730000000000000000000000000000000000000000000000000000000'
   ApplIdentityData : '                      '
   ** Origin Context
   PutApplType    : '6'
   PutApplName    : '                              '
   PutDate  : '19941124'     PutTime  : '11240678'
   ApplOriginData : '     '




****    Message       ****

 length - 188 bytes

00000000:  736E 6F6F 6B65 7220 2020 2020 2020 2020  'snooker         '
00000010:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000020:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000030:  534E 4F4F 4B45 522E 5749 4748 542E 5443  'SNOOKER.WIGHT.TC'
00000040:  5020 2020 2020 2020 2020 2020 2020 2020  'P               '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000060:  0000 0001 0000 0024 0000 0001 0000 0015  '.......>........'
00000070:  0000 0001 0000 0001 0000 0000 0000 0000  '................'
00000080:  0000 0002 0000 0004 0000 0028 0000 0DAD  '...........(....'
00000090:  0000 0000 0000 0014 534E 4F4F 4B45 522E  '........SNOOKER.'
000000A0:  5749 4748 542E 5443 5020 2020 0000 0003  'WIGHT.TCP   ....'
```

```
000000B0:   0000 0010 0000 05E7 0000 0001          '............    '



 MQGET of message number 4
****Message descriptor****

  StrucId  : 'MD '  Version : 1
  Report   : 0  MsgType : 2
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'MQADMIN '
  Priority : 8  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED477D63826C000'
  CorrelId : X'414D5120736E6F6F6B657220202020202ED477D62A9EA100'
  BackoutCount : 0
  ReplyToQ        : '                                          '
  ReplyToQMgr     : 'snooker                                   '
  ** Identity Context
  UserIdentifier : 'tiger        '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                             '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                             '
  PutDate  : '19941124'    PutTime  : '11240694'
  ApplOriginData : '    '



****    Message      ****

 length - 36 bytes

00000000:  0000 0002 0000 0024 0000 0001 0000 0015 '.......>........'
00000010:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000020:  0000 0000                               '....        '



 No more messages
 MQCLOSE
 MQDISC
```

Working with local queues

This section contains examples of some of the MQSC commands that you can use. Refer to the *MQSeries Command Reference* for a complete description of these commands.

This section describes how you use some of these MQSC commands to manage:

- Local queues.

- Queue aliases (also referred to as alias queues).

- Model queues.

- Remote queues (also referred to as a local definition of a remote queue).

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 MQSC command DEFINE QLOCAL 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 MQSC command does this:

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

Notes:

1. Most of these attributes *are* the defaults as supplied with the product. However, they are shown here for purposes of illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying default object attributes".

2. USAGE (NORMAL) indicates that this queue is not a transmission queue.

3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute, if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes".

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 ALTER QMGR 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 in the file `amqscoma.tst`. This queue is automatically created when you run the sample. 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 and its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle.

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 11. "The MQSeries dead-letter queue handler".

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:

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

Note: The syntax of this command is different from that of the corresponding DEFINE command.

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

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

This command displays the three specified attributes as follows:

```
AMQ8409: Display Queue details.
    QUEUE(ORANGE.LOCAL.QUEUE)
    MAXDEPTH(1000)
    MAXMSGL(2000)
    CURDEPTH(0)
```

CURDEPTH is the current queue depth, that is, the number of messages on the queue. This is a

useful attribute to display, because you can use it to monitor the queue depth to ensure that the queue does not become full.

Copying a local queue definition

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

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

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

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

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

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

Notes:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.

2. If you a define a local queue, without specifying LIKE, it is the same as DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).

Changing local queue attributes

You can change queue attributes in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE attribute. In "Defining a local queue", 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 ALTER command:

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

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

- Using the DEFINE command with the REPLACE option, for example:

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

This command changes not only the maximum message length, but 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:

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

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

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

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 TARGQ attribute (*BaseQName* in PCF). When an application specifies an alias queue in an MQI call, the queue manager resolves the real queue name at runtime.

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

Defining an alias queue

The following command creates an alias queue:

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

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

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

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGQ (MAGENTA.QUEUE) REPLACE
```

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

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

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

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

You can do this using the following commands:

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

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

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

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

ALPHA uses the queue name ALPHAS.ALIAS.QUEUE in its MQI calls; BETA uses the queue

name BETAS.ALIAS.QUEUE. They both access the same queue, but in different ways.

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

Using other commands with queue aliases

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

```
* Display the queue alias' attributes
* ALL = Display all attributes

DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE) ALL


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

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


* Delete this queue alias, if you can.

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

You cannot delete a queue alias if, for example, an application currently has the queue open or has a queue open that resolves to this queue. See the *MQSeries Command Reference* manual for more information about this and other queue alias commands.

Working with model queues

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

Model queues provide a convenient method for applications to create queues as 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:

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
       DESCR('Queue for messages from application X') +
       PUT (DISABLED) +
```

```
        GET (ENABLED) +
        NOTRIGGER +
        MSGDLVSQ (FIFO) +
        MAXDEPTH (1000) +
        MAXMSGL (2000) +
        USAGE (NORMAL) +
        DEFTYPE (PERDYN)
```

This command creates a model queue definition. From the DEFTYPE 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 LIKE and REPLACE attributes when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

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

```
* Display the model queue's attributes
* ALL = Display all attributes

DISPLAY QUEUE (GREEN.MODEL.QUEUE) ALL


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

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


* Delete this model queue:

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

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 for IRIX.

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI.

Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC).

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

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

In this example:

QLOCAL (MOTOR.INSURANCE.QUEUE)

The name of the application queue being defined.

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)

The name of the application to be started by a trigger monitor program.

MAXMSGL (2000)

The maximum length of messages on the queue.

DEFPSIST (YES)

The default persistence for messages is that messages are persistent on this queue.

INITQ (MOTOR.INS.INIT.QUEUE)

The name of the initiation queue on which the queue manager is to put the trigger message.

TRIGGER

The trigger attribute value.

TRIGTYPE (DEPTH)

A trigger event is generated when the number of messages of the required priority (TRIMPRI) reaches the number specified in TRIGDPTH.

TRIGDPTH (100)

The number of messages required to generate a trigger event.

TRIGMPRI (5)

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

Defining an initiation queue

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

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
       GET (ENABLED) +
       NOSHARE +
       NOTRIGGER +
       MAXMSGL (2000) +
       MAXDEPTH (10)
```

Creating a process definition

Use the DEFINE PROCESS 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 PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE.

The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

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

MOTOR.INSURANCE.QUOTE.PROCESS

The name of the process definition.

DESCR ('Insurance request message processing')

The text following the keyword is a description of the application program to which the definition relates. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must use the single quotes.

APPLTYPE (UNIX)

The type of the application is one that runs on IRIX

APPLICID ('/u/admin/test/IRMP01.o')

The name of the application executable.

USERDATA ('open, close, 235')

User-defined data, which can be used by the application.

Displaying your process definition

Use the DISPLAY PROCESS command, with the ALL keyword, to examine the results of your definition. For example:

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


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

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

Chapter 7. Administering remote MQSeries objects

This chapter describes how to administer MQSeries objects on another queue manager. It also describes how you can use remote queue objects to control the destination of messages and reply messages.

It contains these sections:

- "Understanding channels and remote queuing"

- "Remote administration"

- "Creating a local definition of a remote queue"

- "Using remote queue definitions for aliasing"

For more information about channels, their attributes, and how to set them up, refer to the *MQSeries Distributed Queuing Guide.*

Understanding channels and remote queuing

Queue managers communicate with each other using channels. For example, if an application is to put a message on a queue managed by a remote queue manager, a channel must be set up between the two queue managers. The channel is defined to the queue managers at each end of the connection. Each channel is named and has a number of attributes that define, for example, the type of channel, the protocol to be used for communication.

Channels are used for sending messages between queue managers. These messages may originate from:

- User-written application programs that transfer data from one node to another.

- User-written administration applications that use PCFs.

- Queue managers sending:

    Instrumentation event messages to another queue manager.

    MQSC commands issued from a **runmqsc** command in indirect mode--where the commands are run on another queue manager.

Channels are unidirectional, that is, messages can only be sent in one direction. Channel definitions are made in complementary pairs, one at each end of the connection. For example, if one end is a sender, the other must be a receiver.

Channels are 'linked' to queue managers (and therefore the applications they serve) by transmission queues and remote queue definitions. A transmission queue is used to forward messages (through a channel) to another queue manager. A remote queue definition identifies a queue on another queue manager. To give you an idea of how these things can fit together:

- A remote queue definition specifies a transmission queue.

- A channel serves a transmission queue, which is specified when the channel is defined.

"Preparing channels and transmission queues for remote administration" shows how to use these definitions to set up remote administration.

You define a channel using the DEFINE CHANNEL MQSC command. Channels, their attributes, and how you use them in distributed queuing, are discussed at length in the *MQSeries Distributed Queuing Guide.* In this section, the examples concerned with channels use the default channel attributes unless otherwise specified.

Creating a local definition of a remote queue

You can use a remote queue definition as a local definition of remote queue. You create a remote queue object--on your local queue manager--to identify a local queue on another queue manager.

Understanding a local definition of a remote queue

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

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

Example of using a local definition of a remote queue

Purpose:

An application is required to put a message on a queue owned by a remote queue manager.

How it works:

The application connects to a queue manager called saturn.queue.manager. The destination queue is owned by another queue manager.

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

Field Value	Description
ObjectName CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the destination queue and the destination queue manager.
ObjectType (Queue)	Identifies this object as queue.
ObjectQmgrName Blank or saturn.queue.manager	This field is optional: If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition was made and to which the application is connected). If not blank, the name of local queue manager must be specified.

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

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

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

Where:

QREMOTE (CYAN.REMOTE.QUEUE)

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

DESCR ('Queue for auto insurance requests from the branches')

Some descriptive text, that you can put in to remind yourself and others what this queue is for.

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)

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

RQMNAME (jupiter.queue.manager)

The name of the remote queue manager that owns the destination queue
AUTOMOBILE.INSURANCE.QUOTE.QUEUE.

XMITQ (INQUOTE.XMIT.QUEUE)

The name of the transmission queue. This is optional, if not specified, a queue with the same name as the remote queue manager is used.

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

Putting messages on a remote queue - another way

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

Using other commands with remote queues

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

```
* Display the remote queue's attributes.
* ALL = Display all attributes

DISPLAY QUEUE (CYAN.REMOTE.QUEUE) ALL


* ALTER the remote queue to enable puts.
* This does not affect the destination queue,
* only applications that specify this remote queue.



ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)


* Delete this remote queue
* This does not affect the destination queue
* only its local definition



DELETE QREMOTE (CYAN.REMOTE.QUEUE)
```

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

Creating a transmission queue

A transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel. The channel provides a one-way link to the remote queue manager. Messages are queued at the transmission queue until the channel can accept them. When you define a channel, you must specify a transmission queue name at the sending end of the message channel.

The *Usage* attribute (USAGE in MQSC) defines whether a queue is a transmission queue or a normal queue.

Default transmission queues

Optionally, you can specify a transmission queue in a remote queue object, using the *XmitQName* attribute (XMITQ in MQSC). If no transmission queue is defined, a default is used. When applications put messages on a remote queue, if a transmission queue with the same name as the destination queue manager exists, that queue is used. If this queue does not exist, the queue specified by the *DefaultXmitQ* attribute (DEFXMITQ in MQSC) on the local queue manager is used.

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

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

Applications can put messages directly on a transmission queue, or they can be put there indirectly, for example, through a remote queue definition. See also "Creating a local definition of a remote queue".

Remote administration

This section tells you how to administer a remote queue manager from a local queue manager. You can implement remote administration from a local node using:

- MQSC commands

- PCF commands

Preparing the queues and channels is essentially the same for both methods. In this book, the examples show MQSC commands, because they are easier to understand. However, you can convert the examples to PCFs if you wish. For more information about writing administration programs using PCFs, see the *MQSeries Programmable System Management*.

In remote administration you send MQSC commands to a remote queue manager--either interactively or from a text file containing the commands. The remote queue manager may be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in different MQSeries environments, including AIX, IRIX, AS/400, MVS/ESA, and

OS/2.

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

Preparing queue managers for remote administration

Figure 13 shows the configuration of queue managers and channels that are required for remote administration. `source.queue.manager` is the *source* queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned, if possible. `target.queue.manager` is the destination queue manager, which processes the commands and generates any operator messages.

Figure 13. Remote administration
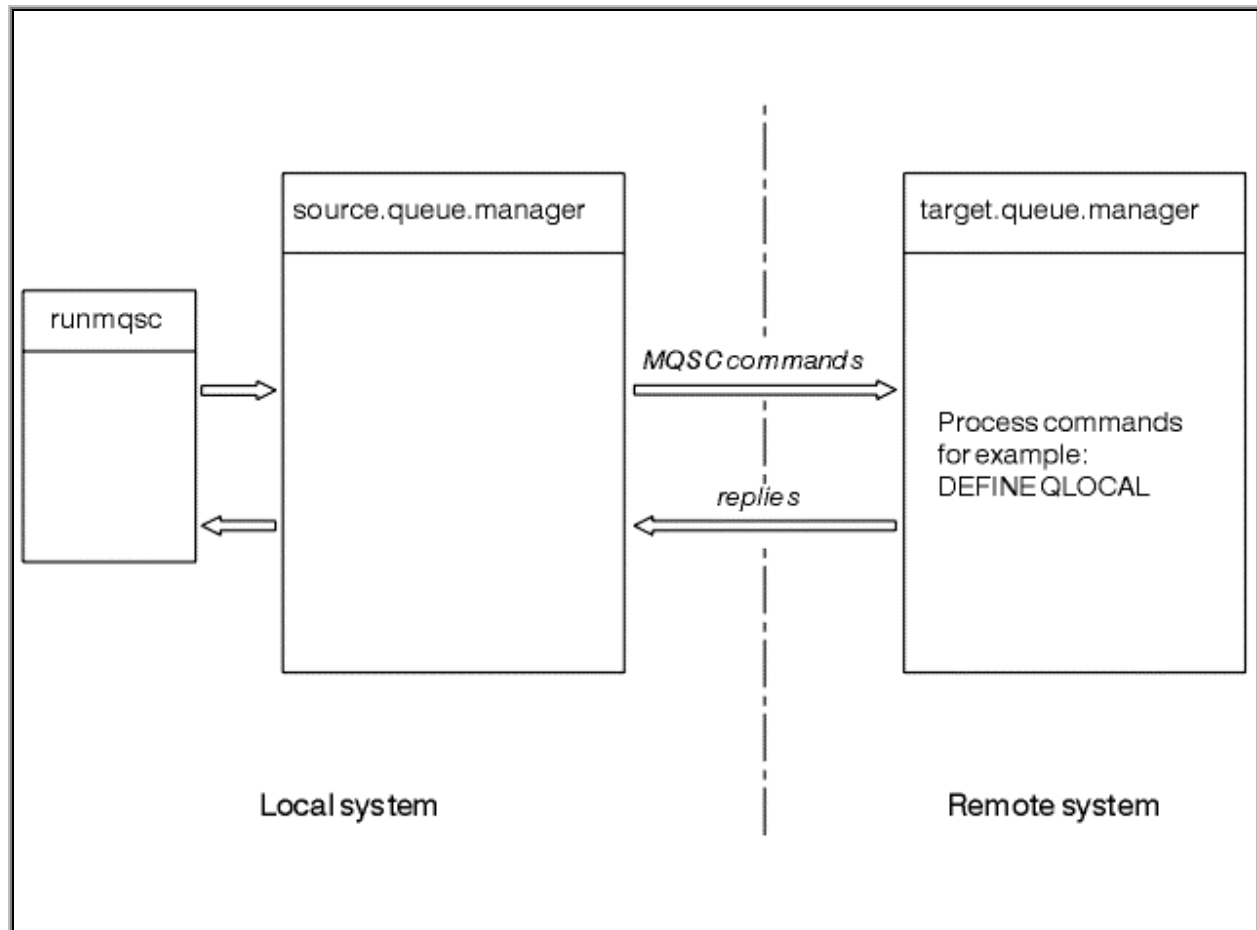
On both systems, if you have not already done so, you must:

- Create the queue manager, using the **crtmqm** command.

- Start the queue manager, using the **strmqm** command.

- Run the sample `amqscoma.tst`, using the **runmqsc** command.

See "Creating the default and system objects" for more information about these steps. You have to run these commands locally or over a network facility, for example, telnet.
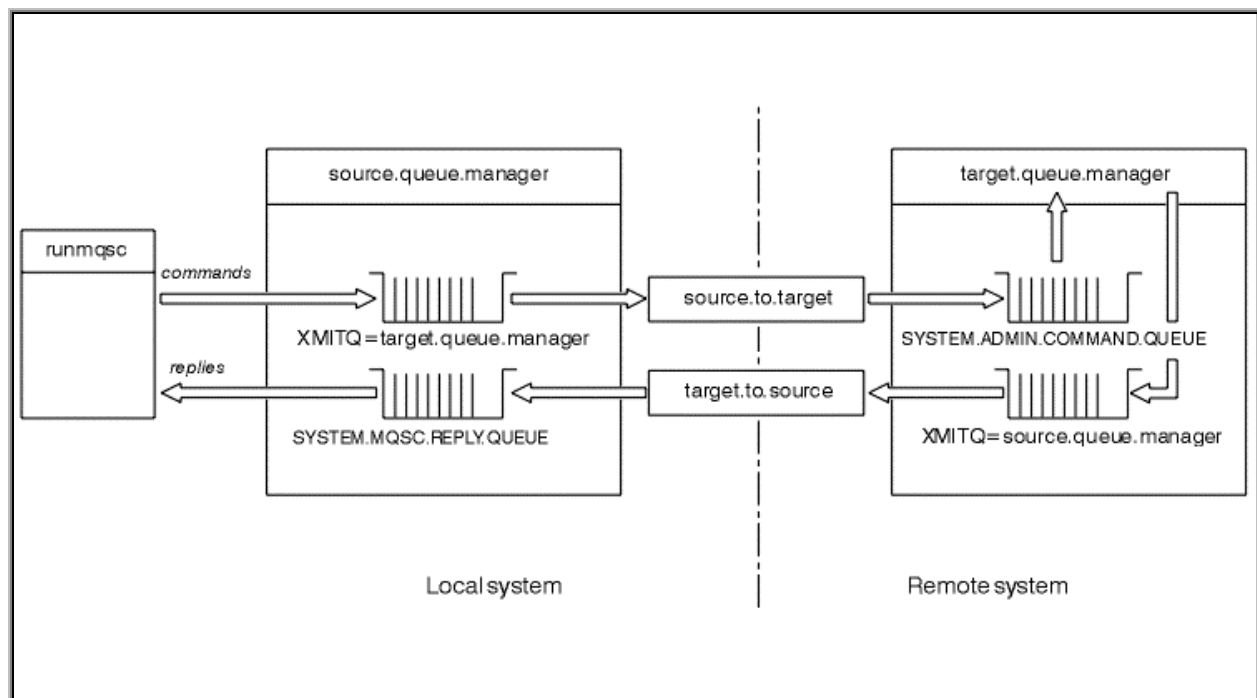
On the destination queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This is created from the sample command file `amqscoma.tst`.

- The command server must be started, using the **strmqcsv** command.

Preparing channels and transmission queues for remote administration

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

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the destination. Its sender is at `source.queue.manager` and its receiver is at queue manager `target.queue.manager`. The channel `target.to.source` is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each sender. This queue is a local queue that is given the name of the receiving queue manager. Figure 14 summarizes this configuration. However, you should be aware that the SYSTEM.MQSC.REPLY.QUEUE is the name of the model queue in AMQSCOMA.TST that is used by MQSC to develop its own dynamic reply queue. This queue name varies and is internal to MQSC.

Figure 14. Setting up channels and queues for remote administration

See the *MQSeries Distributed Queuing Guide* for more information about setting up remote channels.

Defining channels and transmission queues

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

```
 * Define the sender channel at the source queue manager


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

 * Define the receiver channel at the source queue manager

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

 * Define the transmission queue on the source

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

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

```
 * Define the sender channel on the destination queue manager

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

 * Define the receiver channel on the destination queue manager

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

 * Define the transmission queue on the destination queue manager

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

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

Start the channels

The following description assumes that both ends of the channel are running on MQSeries for IRIX. If this is not the case, refer to the relevant documentation for the non-IRIX end of the channel.

To start the two channels, first ensure that the IRIX `inetd` daemons have been configured for MQSeries on both nodes and are running at both ends of the connections.

Then start the channels, again as background processes:

- On the source queue manager, type:

```
runmqchl -c source.to.target &
```

- On the destination queue manager, type:

```
runmqchl -c target.to.source &
```

The **runmqchl** command is an MQSeries for IRIX control command. It cannot be issued using **runmqsc**.

Issuing MQSC commands remotely

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

- On the destination queue manager, type:

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

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

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

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

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

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

At the end of the MQSC session, the local queue manager displays any timed-out responses that have arrived. When the MQSC session is finished, any further responses are discarded.

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

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

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

Working with queue managers on MVS/ESA

You can issue MQSC commands to an MVS/ESA queue manager from an MQSeries for IRIX queue manager. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the -x flag to the **runmqsc** command on an IRIX node:

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

On the sender channel, set the CONVERT attribute to YES. This specifies that the required data conversion between the systems is performed at the IRIX end. The channel definition command now becomes:

```
* Define the sender channel at the source queue manager on IRIX

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

You must also define the receiver channel and the transmission queue at the source queue manager as before. Again, this example assumes that TCP/IP is the transmission protocol being used.

Recommendations for remote queuing

When you are implementing remote queuing:

- Put the MQSC commands to be run on the remote system in a command file.

- Verify your MQSC commands locally, by specifying the -v flag on the **runmqsc** command.

  You cannot use **runmqsc** to verify MQSC commands on another queue manager.

- Check, as far as possible, that the command file runs locally without error.

- And finally, run the command file against the remote system.

If you have problems using MQSC remotely

If you have difficulty in running MQSC commands remotely, check the following list to see if you have:

- Started the command server on the destination queue manager.

- Defined a valid transmission queue.

- Defined the two ends of the message channels for both:

  - The channel along which the commands are being sent.

  - The channel along which the replies are to be returned.

- Specified the correct connection name (CONNAME) in the channel definition.

- Started the listeners before you started the message channels.

- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.

See also "If you have problems with MQSC".

Using remote queue definitions for aliasing

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

- Queue manager aliasing

- Reply-to queue aliasing

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

As usual in remote queuing, the appropriate channels must be set up if the message is to arrive at its destination.

Queue manager aliasing

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

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

For more information about queue manager aliasing, see the *MQSeries Distributed Queuing Guide.*

Reply-to queue aliasing

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue. If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

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

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

For more information about request messages, reply messages, and reply-to queues, see the *MQSeries Application Programming Reference*. For more information about reply-to queue aliasing, see the *MQSeries Distributed Queuing Guide.*

Chapter 8. Security

This chapter describes the features of security control in MQSeries for IRIX and how you can implement this control.

It contains these sections:

- ["Why you need to protect MQSeries resources"](#)

- ["Understanding the Object Authority Manager"](#)

- ["Using the OAM commands"](#)

- ["Object Authority Manager guidelines"](#)

- ["Understanding authorization files"](#)

Before you begin

All queue manager resources run with these IDs:

```
User ID      mqm
Group        mqm
```

You must create this IRIX user ID and group before you can access any queue manager object. For more information about setting the IDs, see ["Preparing for installation"](#).

User IDs in MQSeries for IRIX user group mqm

If your user ID belongs to IRIX group mqm, you have all authorities to all resources. Your user ID **must** belong to IRIX group mqm to be able to use all the MQSeries for IRIX control commands except **crtmqcvx**. In particular, you need this authority to:

- Use the **runmqsc**. command to run MQSC commands.

- Administer authorities on MQSeries for IRIX using the **setmqaut** command.

If you are sending channel commands to queue managers on a remote IRIX system, you must ensure that your user ID is a member of IRIX group mqm on the target system. For a list of PCF and MQSC channel commands, see ["Channel command security"](#).

It is not essential for your user ID to belong to group mqm for issuing:

- PCF commands--including Escape PCFs--from an administration program.

- MQI calls from an application program.

For more information

For more information about:

- MQSeries for IRIX command sets, see .

- MQSeries for IRIX control commands, see .

- PCF commands and Escape PCFs, see the *MQSeries Programmable System Management* manual.

- MQI calls, see the *MQSeries Application Programming Guide* and *MQSeries Application Programming Reference* manuals.

Why you need to protect MQSeries resources

Because MQSeries queue managers handle the transfer of information that is potentially valuable, you need the safeguard of an authority system. This ensures that the resources that a queue manager owns and manages are protected from unauthorized access, which could lead to the loss or disclosure of the information. In a secure system, it is essential that none of the following are accessed or changed by any unauthorized user or application:

- Connections to a queue manager.

- Access to MQSeries objects such as queues, channels, and processes.

- Commands for queue manager administration, including MQSC commands and PCF commands.

- Access to MQSeries messages.

- Context information associated with messages.

You should develop your own policy with respect to which users have access to which resources.

Understanding the Object Authority Manager

By default, access to queue manager resources is controlled through an authorization service installable component. This component is formally called the Object Authority Manager (OAM) for MQSeries for IRIX. It is supplied with MQSeries for IRIX and is automatically installed and enabled for each queue manager you create, unless you specify otherwise. In this chapter, the term OAM is used to denote the Object Authority Manager supplied with this product.

The OAM is an *installable component* of the authorization service. Providing the OAM as an installable service gives you the flexibility to:

- Replace the supplied OAM with your own authorization service component using the

interface provided.

- Augment the facilities supplied by the OAM with those of your own authorization service component, again using the interface provided.

- Remove or disable the OAM and run with no authorization service at all.

For more information on installable services, see the *MQSeries Programmable System Management.*

The OAM manages users' authorizations to manipulate MQSeries objects, such as queues, process definitions, and channels. It also provides a command interface thorough 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.

How the OAM works

The OAM works by exploiting the security features of the underlying IRIX operating system. In particular, the OAM uses IRIX user and group IDs. Users can access queue manager objects only if they have the required authority.

Managing access through user groups

In the command interface, we use the term **principal** rather than user ID. The reason for this is that authorities granted to a user ID can also be granted to other entities, for example, an application program that issues MQI calls, or an administration program that issues PCF commands. In these cases, the principal associated with the program is not necessarily the user ID that was used when the program was started. However, in this discussion, principals and user IDs are always IRIX user IDs.

Group sets and the primary group

Managing access permissions to MQSeries resources is based on IRIX *user groups,* that is, groups of principals. A principal can belong to one or more IRIX groups. If it belongs to more than one group, the groups to which it belongs are known as its group set. One of the groups in the group set is chosen to be the *primary group.*

The OAM maintains authorizations at the level of groups rather than individual principals. The mapping of principals to group names is carried out within the OAM and operations are carried out at the group level. You can, however, display the authorizations of an individual principal.

When a principal belongs to more than one group

The authorizations that a principal has are the union of the authorizations of all the groups of which it is a member, that is, its group set. Whenever a principal requests access to a resource, the OAM computes this union, and then checks the authorization against it. You can use the control command **setmqaut** to set the authorizations for a specific principal, however, this also gives the same authorizations to the principal's primary group.

Note: Changes to a group's membership do not come into effect until a queue manager is reset, that is, stopped and restarted.

The group authorizations associated with a principal are cached when they are computed by the OAM. Any changes made to a group's authorizations after it has been cached are not recognized until the queue manager is restarted. Avoid changing any authorizations while the queue manager is running.

Default user group

The OAM recognizes a default user group to which all users are nominally assigned. This group has a group ID of **nobody**. By default, no authorizations are given to this group. Users without specific authorizations can be granted access to MQSeries resources through this group ID.

Resources you can protect with the OAM

Through OAM you can control:

- Access to MQSeries objects through the MQI. When an application program attempts to access an object, the OAM checks if the user ID making the request has the authorization (through its user group) for the operation requested.

  In particular, this means thar queues, and the messages on queues, can be protected from unauthorized access.

- Permission to use MQSC commands; only members of user group mqm can execute queue manager administration commands, for example, to create a queue.

- Permission to use control commands; only members of user group mqm can execute control commands, for example, creating a queue manager, starting a command server, or using **runmqsc**.

- Permission to use PCF 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 only be allowed 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.

Using groups for authorizations

Using groups, rather than individual principals, for authorization reduces the amount of administration required. Typically, a particular kind of access is required by more than one principal. For example, you might define a group consisting of end users who want to run a particular application. New users can be given access simply by adding the appropriate group to their IRIX user ID.

Try to keep the number of groups as small as possible. For example, dividing principals into one group for application users and one for administrators is a good place to start.

Caution
Remember, when you change the authorization of a principal you also change the authorization of its primary group. This makes it especially important to ensure that you do not change the authorization of a principal inadvertently, simply because it belongs to the same primary group as the principal you specified when you changed an authorization. You can avoid this problem if you 'think groups' right from the start

Disabling the object authority manager

By default, the OAM is enabled. You can disable it by setting the IRIX operating system variable MQSNOAUT before the queue manager is created, as follows:

```
export MQSNOAUT=yes
```

However, if you do this you cannot, in general, restart the OAM later. A much better approach is to have the OAM enabled and ensure that all users and applications have access through an appropriate user ID.

You can also disable the OAM for testing purposes only by removing the authorization service stanza in the queue manager configuration file (qm.ini).

Using the OAM commands

The OAM provides a command interface for granting and revoking authority. Before you can use these commands, you must be suitably authorized--your user ID must belong to the IRIX group mqm. This group should have been set up when you installed the product, see "Preparing for installation".

If your user ID is a member of mqm, you have a 'super user' authority to the queue manager. This means that you are authorized to issue any MQI request or command from your user ID.

The OAM provides two commands that you can invoke from your IRIX shell to manage the authorizations of users. These are:

- **setmqaut** (Set or reset authority)

- **dspmqaut** (Display authority)

Authority checking occurs in the following calls: **MQCONN**, **MQOPEN**, **MQPUT1**, and **MQCLOSE**.

Authority checking is only performed at the first instance of any of these calls, and authority is not amended until you reset (that is, close and reopen) the object.

Therefore, any changes made to the authority of an object, using **setmqaut** do not take effect until you reset the object.

What you specify when you use the OAM commands

The authority commands apply to the specified queue manager; if you do not specify a queue manager, the default queue manager is used. On these commands, you must specify the object uniquely, that is, you must specify the object name and its type. You also have to specify the principal or group name to which the authority applies.

Authorization lists

On the **setmqaut** command you specify a list of authorizations. This is simply a shorthand way of specifying whether authorization is to be granted or revoked, and which resources the authorization applies to. Each authorization in the list is specified as a lowercase keyword, prefixed with a + or - sign. Use a + sign to add the specified authorization or a - sign to remove the authorization. You can specify any number of authorizations in a single command. For example:

```
  +browse -get +put
```

Using the setmqaut command

Provided you have the required authorization, you can use the **setmqaut** command to grant or revoke authorization of a principal or user group to access a particular object. The following example shows how the **setmqaut** command is used:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE -g GroupA
+browse -get +put
```

In this example:

This term...	Specifies the...
saturn.queue.manager	Queue manager name.
queue	Object type.
RED.LOCAL.QUEUE	Object name.
GroupA	ID of the group to be given the authorizations.
+browse -get +put	Authorization list for the specified queue. These must be no spaces between the '+' or '-' signs and the keyword.

The authorization list specifies the authorizations to be given, where:

This term...	Specifies...
+browse	Add authorization to browse (**MQGET** with browse option) messages on the queue.
-get	Remove authorization to get (**MQGET**) messages from the queue.

`+put`   Add authorization to put (**MQPUT**) messages on the queue.

This means that applications started with user IDs that belong to IRIX user group `GroupA` have these authorizations.

You can specify one or more principals and, at the same time, one or more groups. For example, this command revokes put authority on the queue `MyQueue` to the principal `FvUser` and to groups `GroupA` and `GroupB`.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p FvUser -g GroupA -g
GroupB -put
```

Note: This command also revokes put authority for all principals in the primary group of `FvUser`.

For a formal definition of the command and its syntax, see [setmqaut (Set/reset authority)](#).

Authority commands and installable services

The **setmqaut** command takes an additional parameter that specifies the name of the installable service component to which the update applies. You must specify this parameter if you have multiple installable components running at the same time. By default, this is not the case. If the parameter is omitted, the update is made to the first installable service of that type, if one exists. By default, this is the supplied OAM.

Access authorizations

Authorizations defined by the authorization list associated with the **setmqaut** command 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.

Each authorization is specified by a keyword used with the **setmqaut** and **dspmqaut** commands. These are described in [setmqaut (Set/reset authority)](#).

Display authority command

You can use the command **dspmqaut** to view the authorizations that a specific principal or group has for a particular object. The flags have the same meaning as those in the **setmqaut** command. Authorization can only be displayed for one group or principal at a time. See [dspmqaut (Display authority)](#) for a formal specification of this command.

For example, the following command displays the authorizations that the group `GpAdmin` has to a

process definition named `Annuities` on queue manager `QueueMan1`.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

The keywords displayed as a result of this command identify the authorizations that are active.

Object Authority Manager guidelines

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

- Starting and stopping queue managers.

- Accessing certain special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE.

- Programs that use full MQI context options.

- In general, creating and copying application queues.

User IDs

The special user ID `mqm` that you create is intended for use by the product only. It should never be available to non-privileged users.

The user ID that is used for authorization checks is the IRIX logged-in user ID. However, if a process is not logged on, for example, an inetd daemon,   the effective user ID of the process is used.

The creator of an object is the owner of that object. Public access is always NONE unless you change it explicitly.

Queue manager directories

The directory containing queues and other queue manager data is private to the product. Objects in this directory have IRIX user authorizations that relate to their OAM authorizations. However, do not use standard IRIX commands to grant or evoke authorizations to MQI resources because:

- MQSeries objects are not necessarily the same as the corresponding system object name. See "Understanding MQSeries file names" for more information about this.

- All objects are owned by user ID `mqm`.

Queues

The authority to a dynamic queue is based on--but not necessarily the same as--that of the model queue from which it is derived. See the notes following Figure 15 for more information.

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 ID to access to

an alias queue that resolves to a local queue to which the user ID has no access permissions.

You should limit the authority to create queues to privileged users. If you do not, some users may bypass the normal access control simply by creating an alias.

Alternate user authority

Alternate user authority controls whether one user ID can use the authority of another user ID 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 ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.

- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

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

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

Note: You can use alternate user IDs on any MQSeries object. Use of an alternate user ID does not affect the user ID 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. The context information comes in two sections:

Identity section

> This part specifies who the message came from. It consists of the following fields:
>
> - *UserIdentifier*
> - *AccountingToken*
> - *ApplIdentityData*

Origin section

> This section specifies where the message came from, and when it was put onto the queue. It consists of the following fields:
>
> - *PutApplType*

- *PutApplName*

- *PutDate*

- *PutTime*

- *ApplOriginData*

Applications can specify the context data when either an **MQOPEN** or an **MQPUT** call is made. This data may be generated by the application, it may be passed on from another message, or it may be generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application, running under an authorized user ID.

A server program can use the *UserIdentifier* to determine the user ID of an alternate user.

You use context authorization to control whether the user can specify any of the context options on any **MQOPEN** or **MQPUT1** call. For information about the context options, see the *MQSeries Application Programming Guide*. For descriptions of the message descriptor fields relating to context, see the *MQSeries Application Programming Reference* manual.

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 ID. This is the user ID that the message channel agent is running under.

CTX

> The user ID 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, see Figure 15.

Channel exits

> Channel exits can be used for added security.

For more information, see the *MQSeries Distributed Queuing Guide*.

Channel command security

Channel commands can be issued as PCF commands, 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 IRIX system. The user ID, as specified in the message descriptor of the PCF message, must belong to group mqm on the target system. These commands are:

- *ChangeChannel*

- *CopyChannel*

- *CreateChannel*

- *DeleteChannel*

- *PingChannel*

- *ResetChannel*

- *StartChannel*

- *StartChannelInitiator*

- *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 IRIX system either by sending the command directly in a PCF escape message or by issuing the command using **runmqsc** in indirect mode. The user ID as specified in the message descriptor of the associated PCF message must belong to group mqm on the target system. (PCF commands are implicit in MQSC commands issued from **runmqsc** in indirect mode.) These commands are:

- ALTER CHANNEL

- DEFINE CHANNEL

- DELETE CHANNEL

- PING CHANNEL

- RESET CHANNEL

- START CHANNEL

- START CHINIT

- STOP CHANNEL

- RESOLVE CHANNEL

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

Control commands for channels

For the control commands for channels, the user ID that issues them must belong to user group mqm. These commands are:

- **runmqchi** (Run channel initiator)

- **runmqchl** (Run channel)

Understanding the authorization specification tables

The authorization specification tables starting in topic Figure 15 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.

- Adminstration programs that issue PCF commands.

In this section, the information is presented as a set of tables that specify the following:

Action to be performed

> MQI option, MQSC command, or PCF command.

Access control object

> Queue, process, 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 **setmqaut** command 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. See "What the authorization files contain" for more information.

MQI authorizations

An application is only allowed to issue certain MQI calls and options 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.

summarizes the authorizations needed for each call.

Figure 15. Security authorization needed for MQI calls

	Authorization required		
Access control object	**Queue object (10)**	**Process object**	**Queue manager object**
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT
MQOPEN Option			
MQOO_INQUIRE	MQZAO_INQUIRE (1)	MQZAO_INQUIRE (1)	MQZAO_INQUIRE (1)
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT (2)	MQZAO_INPUT	Not applicable	No check
MQOO_OUTPUT (Normal queue) (3)	MQZAO_OUTPUT	Not applicable	No check
MQOO_PASS_ IDENTITY_CONTEXT (4)	MQZAO_PASS_ IDENTITY_ CONTEXT	Not applicable	No check
MQOO_PASS_ ALL_CONTEXT (4,5)	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check

MQOO_SET_ IDENTITY_CONTEXT (4,5)	MQZAO_SET_ IDENTITY_ CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_ CONTEXT (11)
MQOO_SET_ ALL_CONTEXT (4,6)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (11)
MQOO_OUTPUT (Transmission queue) (7)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (11)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	(8)	(8)	MQZAO_ALTERNAT E_ USER_ AUTHORITY (8,9)
MQPUT1 Option			
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_ CONTEXT (13)	Not applicable	No check
MQPMO_PASS_ ALL_CONTEXT	MQZAO_PASS_ ALL_CONTEXT (13)	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_ CONTEXT (13)	Not applicable	MQZAO_SET_ IDENTITY_ CONTEXT (11)
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT (13)	Not applicable	MQZAO_SET_ ALL_CONTEXT (11)
(Transmission queue) (7)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (11)
MQPMO_ALTERNATE _ USER_AUTHORITY	(12)	Not applicable	MQZAO_ALTERNAT E_ USER_ AUTHORITY (9)
MQCLOSE Option			
MQCO_DELETE	MQZAO_DELET E (14)	Not applicable	Not applicable
MQCO_DELETE_PUR GE	MQZAO_DELET E (14)	Not applicable	Not applicable

Specific notes:

(1) Either the queue, process or queue-manager object is checked, depending on the type of object being opened.

(2) MQOO_INPUT_* must be specified as well. This is valid for a local, model, or alias queue.

(3) This check is done for all output cases except that documented in note (7).

(4) MQOO_OUTPUT must also be specified.

(5) MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.

(6) MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.

(7) This check is done 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).

(8) 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.

(9) This authorization allows **any** *AlternateUserId* to be specified.

(10) If a model queue is being opened:

- MQZAO_DISPLAY authority is needed for the model queue, as well as whatever other authorities (also for the model queue) are required for the open options specified.

- 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.

(11) This authority is required for the queue-manager object, as well as for the particular queue.

(12) The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.

(13) An MQZAO_OUTPUT check is also carried out, if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.

(14) The check is carried out only if both of the following are true:

- A permanent dynamic queue is being closed and deleted, and

- 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

Figure 16 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

Figure 16. MQSC commands and security authorization needed (1)

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

Specific notes:

(1) The user identifier, under which the program (for example, runmqsc) which submits the command is running, must also have MQZAO_CONNECT authority to the queue manager.

(2) Either the queue, process 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 **setmqaut** 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

Figure 17 summarizes the authorizations needed for each PCF command.

Figure 17. PCF commands and security authorization needed (1)

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

Specific notes:

(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 admin queue for output.

(2) Either the queue, process 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. Two separate authorities can be granted, one for creating queues (of any name), and one for creating processes (also of any name). These authorities are granted for the object type (queue or process), and a blank object name.

(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 does in fact already exist. 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 object-specific.

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.

Understanding authorization files

The information in this section is given for problem determination. Under normal circumstances, use authorization commands to view and change authorization information.

MQSeries for IRIX uses an specific file structure to implement security. You do not have to do anything with these files, except to ensure that all the authorization files are themselves secure.

Security is implemented by authorization files. From this perspective, there are three types of authorization:

• Authorizations applying to single object, for example, the authority to put a message on an queue.

• Authorizations applying to a class of objects, for example, the authority to create a queue.

• Authorizations applying across all classes of objects, for example, the authority to perform

operations on behalf of different user.

Authorization file paths

The path to an authorization file depends on its type. When you specify an authorization for an object, for example, the queue manager creates the appropriate authorization files. It puts these files into a sub-directory, the path of which is defined by the queue manager name, the type of authorization, and where appropriate, the object name.

Not all authorizations apply directly to instances of objects. For example, the authorization to create an object applies to the class of objects rather than to an individual instance. Also, some authorizations apply across the entire queue manager, for example, alternate user authority means that a user can assume the authorities associated with another user.

Authorization directories

By default, the authorization directories, for a queue manager called saturn are:

/var/mqm/qmgrs/saturn/auth/queues

> Authorization files for queues.

/var/mqm/qmgrs/saturn/auth/procdef

> Authorization files for process definitions.

/var/mqm/qmgrs/saturn/auth/qmanager

> Authorization files for the queue manager.

/var/mqm/qmgrs/saturn/auth/@aclass

> Authorizations applying to all classes.

In the object directories, the @class files hold the authorizations related to the entire class.

Note: There is a difference between @class (the authorization file that specifies authorization for a particular class) and @aclass (the directory that contains a file that specifies authorizations to all classes)

The paths of the object authorization files are based on those of the object itself, where **auth** is inserted ahead of the object type directory. You can use the **dspmqfls** command to display the path to a specified object.

For example, if the name and path of SYSTEM.DEFAULT.LOCAL.QUEUE is:

```
/var/mqm/qmgrs/saturn/queues/SYSTEM!DEFAULT!LOCAL!QUEUE
```

The name and path of the corresponding authorization file is:

```
/var/mqm/qmgrs/saturn/auth/queues/SYSTEM!DEFAULT!LOCAL!QUEUE
```

Note: In this case, the actual names of the files associated with the queue are not the same as the name of the queue itself. See <u>"Understanding MQSeries file names"</u> for details.

What the authorization files contain

The authorizations of a particular group are defined by a set of stanzas in the authorization file. See "Understanding authorization files" for more information. The authorizations apply to the object associated with this file. For example:

```
groupB:
    Authority=0x0040007
```

This stanza defines the authority for the group **groupB**. The authority specification is the union of the individual bit patterns based on the following assignments:

```
Authorization            Formal name                          Hexadecimal
keyword                                                       Value

connect                  MQZAO_CONNECT                        0x00000001
browse                   MQZAO_BROWSE                         0x00000002
get                      MQZAO_INPUT                          0x00000004
put                      MQZAO_OUTPUT                         0x00000008
inq                      MQZAO_INQUIRE                        0x00000010
set                      MQZAO_SET                            0x00000020
passid                   MQZAO_PASS_IDENTITY_CONTEXT          0x00000040
passall                  MQZAO_PASS_ALL_CONTEXT               0x00000080
setid                    MQZAO_SET_IDENTITY_CONTEXT           0x00000100
setall                   MQZAO_SET_ALL_CONTEXT                0x00000200
altusr                   MQZAO_ALTERNATE_USER_AUTHORITY       0x00000400
allmqi                   MQZAO_ALL_MQI                        0x000007FF
crt                      MQZAO_CREATE                         0x00010000
dlt                      MQZAO_DELETE                         0x00020000
dsp                      MQZAO_DISPLAY                        0x00040000
chg                      MQZAO_CHANGE                         0x00080000
clr                      MQZAO_CLEAR                          0x00100000
chgaut                   MQZAO_AUTHORIZE                      0x00800000
alladm                   MQZAO_ALL_ADMIN                      0x009E0000
none                     MQZAO_NONE                           0x00000000
all                      MQZAO_ALL                            0x009E07FF
```

These definitions are made in the header file cmqzc.h In this example, **groupB** has been granted authorizations based on the hexadecimal number **0x40007**. This corresponds to:

```
        MQZAO_CONNECT                    0x00000001
        MQZAO_BROWSE                     0x00000002
        MQZAO_INPUT                      0x00000004
```

```
     MQZAO_DISPLAY                           0x00040000
                                             ----------
     Authority is:                           0x00040007
```

These access rights mean that anyone in groupB can issue the MQI calls:

MQCONN
MQGET (with browse)
MQPUT

and has DISPLAY authority for the object associated with this authorization file.

Class authorization files

The **class authorization files** hold authorizations that relate to the entire class. These files are called **@class** and exist in the same directory as the files for specific objects. The entry MQZAO_CRT in the @class file gives authorization to create an object in the class. This is the only class authority.

All class authorization files.

The **all class authorization file** holds authorizations that apply to an entire queue manager. This file is called **@aclass** and exists in the **auth** subdirectory of the queue manager.

The following authorizations apply to the entire queue manager and are held in the all class authorization file.

entry...

> **Gives authorization to...**

MQZAO_ALTUSR

> Assume the identity of another user when interacting with MQSeries objects.

MQZAO_SET_ALL_CONTEXT

> Set the context of a message when issuing MQPUT.

MQZAO_SET_IDENTITY_CONTEXT

> Set the identity context of a message when issuing MQPUT.

Managing authorization files

Here are some pointers that you need to take into consideration when managing your authorization files:

1. You must ensure that the authorization files are secure and not write-accessable by non-trusted general users. See "Authorizations to authorization files".

2. To be able to reproduce your file authorizations, ensure that you do at least one of the

following:

- Back up the **auth** subdirectory after any significant updates

- Retain shell scripts containing the commands used.

3. Authorization files can be copied and edited. However, these files should not normally have to be created or repaired manually. Should an emergency occur, the information given here can be used to recover lost or damaged authorization files, if required.

Authorizations to authorization files

Authorization files must be readable by any principal. However, only the mqm user ID and the mqm group should be allowed to update these files.

The permissions on authorization files, created by the OAM, are:

```
-rw-rw-r--        mqm        mqm
```

Do not alter these permissions without reviewing carefully whether there are any security exposures.

To alter authorizations using the command supplied with MQSeries for IRIX, your IRIX user ID must either be mqm or it must belong to the mqm group.

Chapter 9. Instrumentation events

You can use the 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 section on instrumentation events in the *MQSeries Programmable System Management* manual.

What instrumentation events are

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

- Queue Full events are enabled for a specified queue.

- 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 threshold limit for the number of messages on a queue is reached.

- A queue is not serviced within a specified time period.

- A channel instance is started or stopped.

- In MQSeries for a UNIX system, an application attempts to open a queue 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 18. 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.

Queue Manager

1. Event conditions

For example:
Queue full
+ event enabled

Event message

2. Event message put on event queue

Event queue

3. Event message processed by a user application

User Application
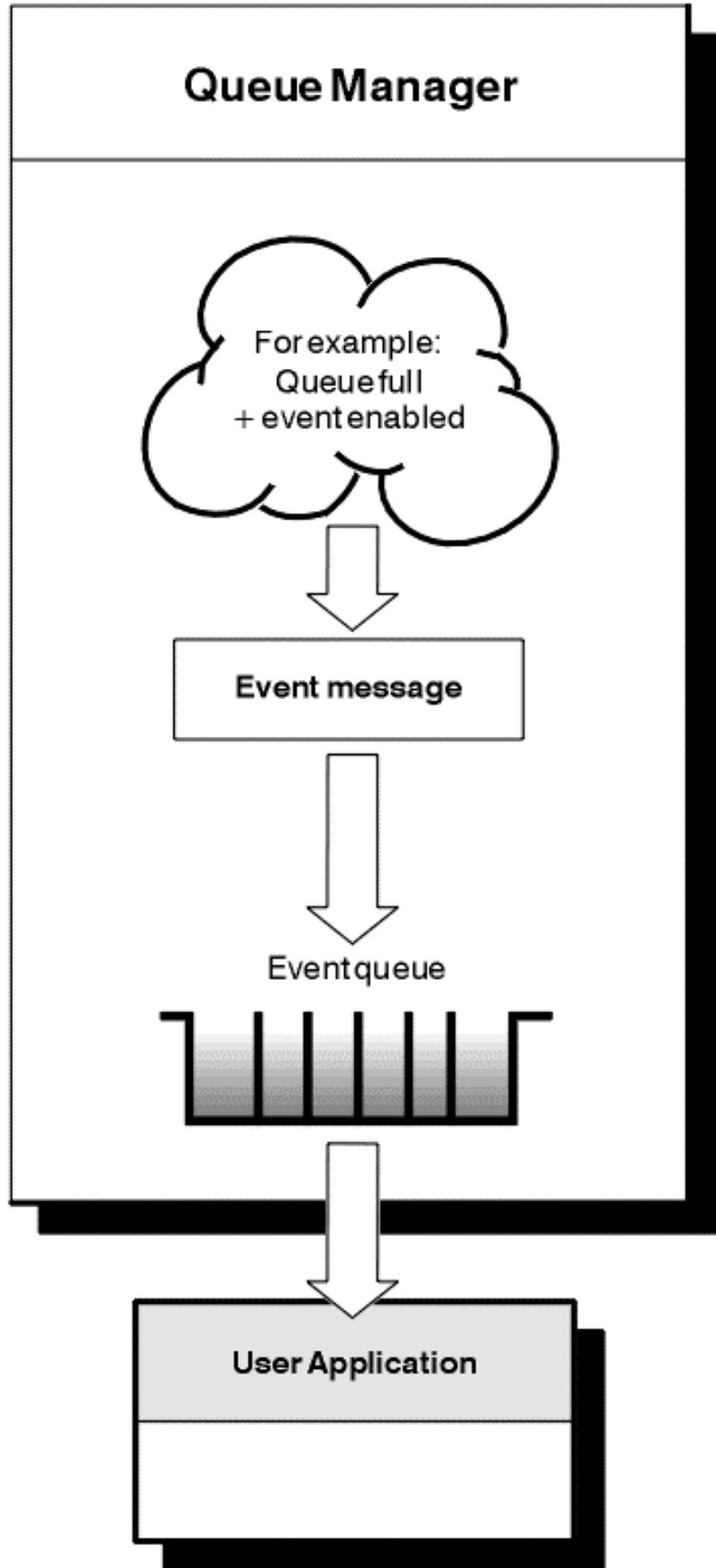
The event message, which contains information about the conditions giving rise to the event, is put onto an *event queue*. An application can retrieve the event message from this queue for analysis.

Why use events?

If you specify your event queues as a 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 19 illustrates this.

Figure 19. Monitoring queue managers across different platforms, on a single node
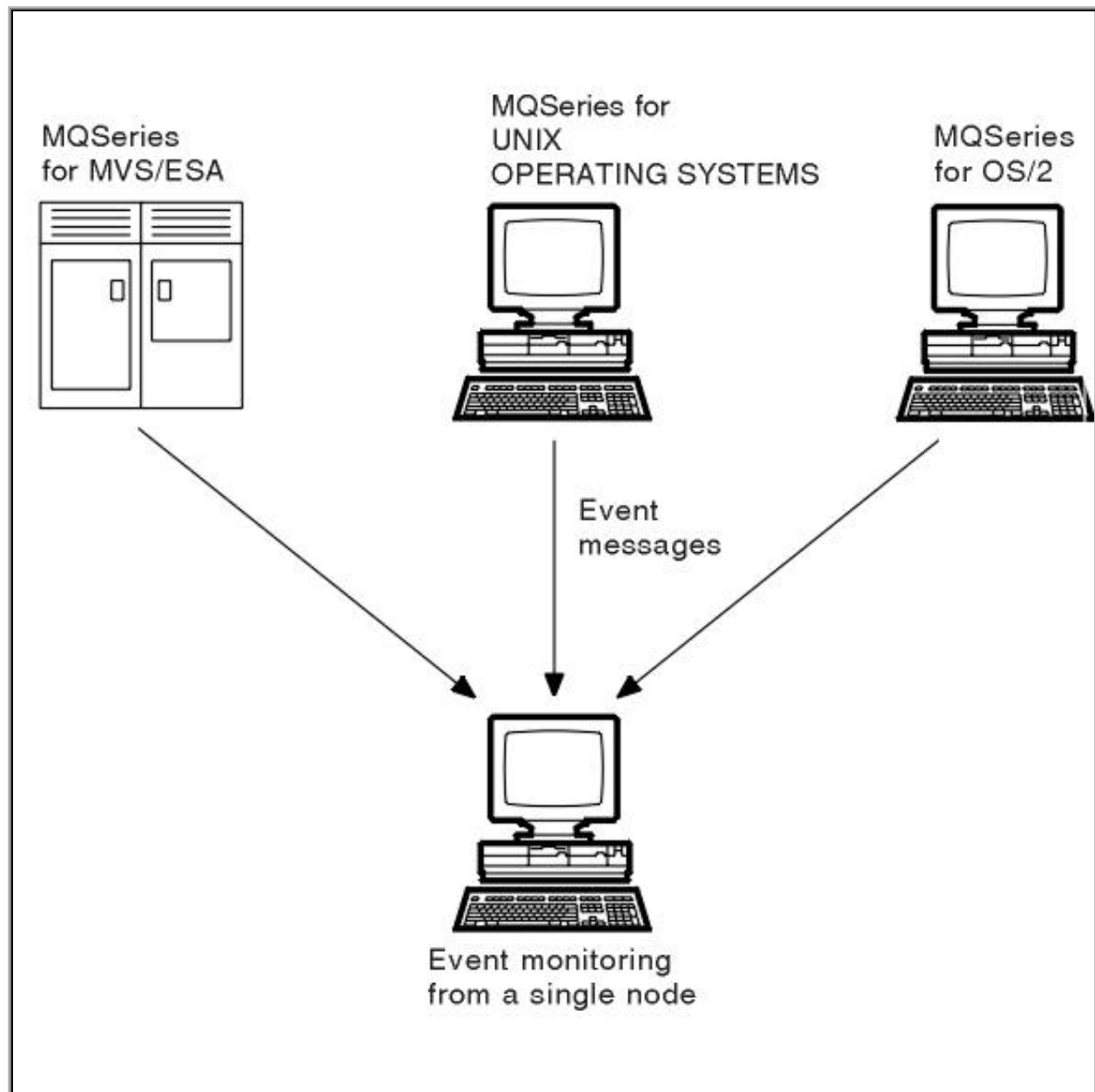
Types of events

MQSeries events may be categorized as follows:

Queue manager events

> These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

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 a get, the queue was not serviced within a predefined time limit.

Channel events

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

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 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* manual.

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, certain events may require that an operator be informed, other events may 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 either of the following:

- MQSC commands. For more information, see the *MQSeries Command Reference* manual.

- PCF commands for queue managers on UNIX systems and OS/2. For more information, see the *MQSeries Programmable System Management* manual.

Enabling an event depends on the category of the event:

- Queue manager events are enabled by setting attributes on the queue manager.

- Performance events as a whole must be enabled on the queue manager, or no performance events can occur. You then enable the specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event, for example, a queue depth high limit.

- Channel events occur automatically; they do not need to be enabled. If you do not want to monitor channel events, you can put-inhibit 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* manual.

Chapter 10. Transactional support and messaging

In applications that use the MQI, put and get operations can be performed under syncpoint control. In MQSeries for IRIX, there are two methods for initiating commit and rollback:

- Using the MQI calls **MQCMIT** and **MQBACK**. This provides a single-phase commit facility, with only the local queue manager involved.

- Using an XA-compliant external syncpoint coordinator, for example, TUXEDO, providing a full two-phase commit. MQSeries for IRIX supports the external syncpoint coordinators through the X/Open XA interface. This means that MQI calls can be included in a transaction where queue manager resources are committed or rolled back as directed by the external coordinator.

Notes:

1. If an external coordinator is used, the **MQCMIT** and **MQBACK** calls become unavailable.

2. Some XA transaction managers require that each XA resource manager supplies its name. This is the string called name in the XA switch structure. The MQSeries for IRIX resource manager name is "MQSeries_XA_RMI".

Interfaces to external syncpoint coordinators

In an XA configuration, MQSeries for IRIX is an XA resource manager that works like other XA resource managers, for example, a database. An XA syncpoint coordinator can manage a set of XA resource managers, and synchronize commit or rollback of transactions in each resource manager. This is how it works:

1. An application informs the syncpoint coordinator that it wants to start a transaction.

2. The syncpoint coordinator informs the resource managers about the current transaction.

3. The application issues calls (for example, **MQGET** in syncpoint) to the resource managers that are associated with the current transaction.

4. The application requests the syncpoint coordinator to commit or rollback the transaction.

5. The syncpoint coordinator completes the transaction by issuing the appropriate calls to each resource manager using 2-phase commit protocols.

XA requires each resource manager to provide a structure called an **XA Switch**. This structure declares the capabilities of the resource manager, and the functions which are to be called by the syncpoint coordinator. This structure is located in the following library:

```
libmqmxa.a
```

IRIX XA library

MQSeries for IRIX supplies two versions of this structure, both of which are contained in the library:

1. *MQRMIXASwitch* for static XA resource management.

2. *MQRMIXASwitchDynamic* for dynamic XA resource management.

The method used to link the structure to a specific XA syncpoint coordinator is defined by the coordinator. For details, consult the documentation supplied with the XA syncpoint coordinator.

The **xa_info** structure that is passed on any **xa_open** call by the syncpoint coordinator specifies the name of the queue manager that is to be administered. This takes the same form as the queue manager name passed to MQCONN, and may be blank if the default queue manager is to be used.

The following restrictions apply:

1. Only one queue manager may be administered by an external syncpoint coordinator at a time. The coordinator has an effective connection to each queue manager, and is therefore subject to the rule that only one connection is allowed at a time.

2. All applications that are run using the syncpoint coordinator can only connect to the queue manager that is administered by the coordinator. They are already effectively connected to that queue manager. They must issue an **MQCONN** to obtain a connection handle and should issue **MQDISC** before they exit.

3. The queue manager administered by an external syncpoint coordinator should be started before the coordinator starts. The syncpoint coordinator should be ended before the queue manager is ended.

Abnormal termination of XA syncpoint coordinator

Because of the nature of the XA interface, abnormal termination of some XA syncpoint coordinators can result in uncommitted transaction branches started before the coordinator terminated, remaining active in MQSeries once the coordinator has been restarted.

The symptom of this is that there may be uncommitted message operations performed under these transaction branches that cannot be resolved by normal means.

If you are using an XA syncpoint coordinator that terminates abnormally, you should stop and restart your queue manager before restarting the syncpoint coordinator to resolve any such transaction branches.

See the *MQSeries Application Programming Guide* for further information on using TUXEDO as your syncpoint cooordinator.

Chapter 11. 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. <u>(1)</u>

Messages can be put on the DLQ by queue managers, by message channel agents (MCAs), and by applications. 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; applications putting messages on the DLQ are strongly recommended to supply an MQDLH. 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 **runmqdlq** command. Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table: when a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

This chapter begins with a description of the DLQ handler, then describes the contents and syntax of the rules table. The chapter concludes with an example rules table.

Invoking the DLQ handler

You invoke the DLQ handler using the **runmqdlq** command. You can supply the name of the DLQ you want to process and the name of that queue's manager as parameters to **runmqdlq**. Alternatively, you can include those values in the rules table. (However, specifying the names of the DLQ and its queue manager as parameters to **runmqdlq** makes it possible for a single rules table to be used for multiple DLQs.) The **runmqdlq** command takes its input from `stdin`: you associate the rules table with **runmqdlq** by redirecting `stdin` from the rules table.

In order to run the DLQ handler, you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. Furthermore, if the DLQ handler is to be able to put messages on queues with the authority of the user ID in the message context, you must be authorized to assume the identity of other users.

For more information about the **runmqdlq** command, see <u>runmqdlq (Run dead-letter queue handler)</u>.

The sample DLQ handler, AMQSDLQ

In addition to the DLQ handler invoked using the **runmqdlq** command, MQSeries provides the source of a sample DLQ handler, AMQSDLQ, whose function is similar to that provided via **runmqdlq**. You can customize AMQSDLQ to provide a DLQ handler that meets specific, local

requirements. For example, you might decide that you want a DLQ handler that can process messages without dead-letter headers. (Both the default DLQ handler and the sample, AMQSDLQ, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

The source of AMQSDLQ is supplied in the directory:

```
C:\MQM\TOOLS\C\SAMPLES\DLQ
```

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

Figure 20 shows an example control-data entry in a DLQ handler rules table.

Figure 20. An example control-data entry in the DLQ handler rules table. The rules table in which this control-data entry appears applies to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

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

The keywords that you can include in a control-data entry are described in the remainder of this section. Please note the following:

- The default value for a keyword, if any, is underlined.

- The vertical line (|) separates alternatives, only one of which can be specified.

- All keywords are optional.

INPUTQ (*QueueName*|'_')

Is the name of the DLQ to which this rules table applies.

INPUTQ (' '), which is the default value, causes the name of the DLQ belonging to the queue manager whose name is supplied as a parameter to the **runmqdlq** command to be used. If no

queue manager name is supplied as a parameter to **runmqdlq**, the DLQ belonging to the queue manager named on the INPUTQM keyword is used. An INPUTQ value specified as a parameter to the **runmqdlq** command overrides any INPUTQ value in the rules table.

INPUTQM (*QueueManagerName*|'_')

Is the name of the queue manager that owns the DLQ named on the INPUTQ keyword. INPUTQM (' '), which is the default value, requests that the default queue manager for the installation be used. An INPUTQM value specified as a parameter to the **runmqdlq** command overrides any INPUTQM value in the rules table.

RETRYINT (*Interval*|60)

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*)

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

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.

As an alternative to including control data in the rules table, you can supply the names of the DLQ and its queue manager as input parameters of the **runmqdlq** command. If any value is specified both in the rules table and on input to the **runmqdlq** command, the value specified on the **runmqdlq** 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 21 shows an example rule from a DLQ handler rules table.

Figure **21. 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.

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

All keywords that you can use on a rule are described in the remainder of this section. 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, only one of which can be specified.

- All keywords except ACTION are optional.

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

The pattern-matching keywords

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

APPLIDAT (*ApplIdentityData*|*)

Is the *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName*|*)

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*|*)

Is the *PutApplType* value specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName*|*)

Is the name of the message queue for which the message is destined.

DESTQM (*QueueManagerName*|*)

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

FEEDBACK (*Feedback*|*)

When the *MsgType* value is MQFB_REPORT, *Feedback* describes the nature of the report.

Symbolic names can be used. 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*|*)

Is the name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType*|*)

Is the message type of the message on the DLQ.

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

PERSIST (*Persistence*|*)

Is the persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

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

REASON (*ReasonCode*|*)

Is the reason code that describes why the message was put to the DLQ.

Symbolic names can be used. 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*|*)

Is the name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName*|*)

Is the name of the queue manager of the reply-to queue, as specified in the message descriptor, MQMD, of the message on the DLQ.

USERID (*UserIdentifier*|*)

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, which you use to describe how a matching message is to be processed, are described below.

ACTION (DISCARD|IGNORE|RETRY|FWD)

Is the action to be taken for any message on the DLQ that matches the pattern defined in this

rule.

IGNORE

> Causes the message to be left on the DLQ.

DISCARD

> Causes the message to be deleted from 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.

The ACTION keyword must be specified. The number of attempts made to implement an action is governed by the RETRY keyword. The interval between attempts is controlled by the RETRYINT keyword of the control data.

FWDQ (*QueueName*|&DESTQ|&REPLYQ)

Is the name of the message queue to which the message should be forwarded when ACTION (FWD) is requested.

QueueName

> Is the name of a message queue. FWDQ(' ') is not valid.

&DESTQ

> Causes the queue name to be taken from the *DestQName* field in the MQDLH structure.

&REPLYQ

> Causes the name to be taken from the *ReplyToQ* field in the message descriptor, MQMD.

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

FWDQM (*QueueManagerName*|&DESTQM|&REPLYQM|'_')

Identifies the queue manager of the queue to which a message is to be forwarded.

QueueManagerName

> Is the name of the queue manager of the queue to which a message is to be forwarded when ACTION (FWD) is requested.

&DESTQM

Causes the queue manager name to be taken from the `DestQMgrName` field in the MQDLH structure.

&REPLYQM

Causes the name to be taken 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

Causes messages to be put 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. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (RetryCount|1)

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 the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Rules table conventions

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

- A rules table must contain at least one rule.

- Keywords can occur in any order.

- A keyword can be included 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.

- 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.

– Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.

– Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.

– Numeric parameters cannot include the question mark (?) wildcard character. The asterisk (*) can be used in place of an entire numeric parameter, but 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. Symbolic names can be used 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.

How the rules table is processed

The DLQ handler searches the rules table for a rule whose pattern matches a message on the DLQ.

The search begins with the first rule in the table, and continues sequentially through the table. When a rule with a matching pattern is found, the action from that rule is attempted. 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.

Please note the following:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.

2. All pattern keywords can be allowed to default, such that a rule 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 is started, and errors are flagged at that time. (Error messages issued by the DLQ handler are described in [Appendix G. "Messages"](#).) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.

4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.

5. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.

6. 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 has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ will be seen, even if the DLQ is defined as first-in-first-out (FIFO). Therefore, if the queue is not empty, a periodic rescan of the DLQ is performed 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 action 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

Figure 22 shows an example rules table that contains a single control-data entry and several rules.

Figure 22. An example rules table for the DLQ handler

```
*
************************************************************************
*         An example rules table for the runmqdlq command            *
************************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
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 organisation
* where the problem is investigated.

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


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

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)


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

```
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

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

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

Chapter 12. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. This means that it must provide a method of tracking the messages in the system, and of recovering messages if the system fails for any reason.

MQSeries ensures that messages are not lost by maintaining records (logs) of the activities of the queue managers that handle the receipt, transmission, and delivery of messages. It uses these logs for three types of recovery:

1. Restart recovery, when you stop MQSeries in a planned way.

2. Crash recovery, when MQSeries is stopped by an unexpected failure.

3. Media recovery, to restore damaged objects.

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped. Any in-flight transactions are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; non-persistent messages are lost during the process.

And that is recovery in a nutshell. The rest of this chapter introduces the concepts of recovery and restart in more detail and then tells you how to recover if problems occur. It covers the following topics:

- "Making sure that messages are not lost (logging)".

- "Using the log for recovery".

Making sure that messages are not lost (logging)

MQSeries records all significant changes to the data controlled by the queue manager in a log. This includes the creation and deletion of objects, all persistent message updates, transaction states, changes to object attributes, and channel activities. Therefore, the log contains the information you need to recover all updates to message queues by:

- Keeping records of queue manager changes.

- Keeping records of queue updates for use by the restart process.

- Enabling you to restore data after a hardware or software failure.

This section tells you more about logs, including:

- "What logs look like"

- "Types of logging"

- "Checkpointing--ensuring complete recovery"

- "Media recovery"

- "Managing logs"

- "Managing log files"

What logs look like

An MQSeries log consists of two components:

1. One or more files of log data

2. A log control file

There are a number of log files which contain the data being recorded. You can define the number and size (as explained in Chapter 13. "Understanding configuration files"), or take the system default of 3 files, each 4MB in size.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used. If you have not changed the log path, they are created in the directory:

```
/var/mqm/log/QmName
```

MQSeries starts with these primary log files, but, if the log starts to get full, allocates *secondary* log files. It does this dynamically, and removes them when the demand for log space reduces. By default, up to 2 secondary log files can be allocated, providing a further 8MB of disk space. The default number can also be changed, see Chapter 13. "Understanding configuration files".

The log control file contains the information needed to monitor the use of log files: their size and location, the name of the next available file, and so on.

Note: You should ensure that the logs created when you start a queue manager are large enough to accommodate the size and volume of messages that your applications will handle. The default log numbers and sizes will require modification to meet your requirements. How to change the default values is described on page "Configuring the logs".

Types of logging

In MQSeries, the number of files that are used for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities:

Circular logging

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are filled. It then goes back to the first file in the ring and starts again. This continues as long as the product is in use and has the advantage that you never run out of log files.

The above is a simple explanation of circular logging. However, there is a complication. The log entries required to restart the queue manager without loss of data are kept until no longer required to ensure queue manager data recovery. The mechanism for releasing log files for reuse is described in "Checkpointing--ensuring complete recovery". For now, you should know that MQSeries uses secondary log files to extend the log capacity as necessary.

You use circular logging if all you want is restart recovery, using the log to roll back transactions that were in progress when the system stopped.

Linear logging

Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so you can always retrieve any record logged from the time that the queue manager was created.

As disk space is finite, you may have to think about some form of archiving. It is an administrative task to manage your disk space for the log, reusing or extending the existing space as necessary.

The number of log files used with linear logging can be very large depending on your message flow and the age of your queue manager. However, there are a number of files which are said to be active. Active files contain the log entries required to restart the queue manager. The number of active log files is usually the same as the number of primary log files as defined in the configuration files. (See Chapter 13. "Understanding configuration files" for further details of how to define the number.)

The key event that controls whether a log file is termed active or not is a *checkpoint*. An MQSeries checkpoint is a group of log records containing information to allow a successful restart of the queue manager. Any information recorded previously is not required to restart the queue manager and can therefore be termed inactive. (See "Checkpointing--ensuring complete recovery" for further information about checkpointing.)

You must decide when inactive log files are no longer required. You may select to archive them, or you may delete them as being no longer of interest to your operation. Refer to "Managing logs" for further information about the disposition of log files.

If a new checkpoint is recorded in the second, or later, primary log file, then the first file becomes inactive and a new primary file is formatted and added to the end of the primary pool, restoring the number of primary files available for logging. In this way the primary log file pool can be seen to be a current set of files in an ever extending list of log files. Again, it is an administrative task to manage the inactive files according to the requirements of your operation.

Although secondary log files are defined for linear logging, they are not used in normal operation. If a situation should arise when, probably due to long lived transactions, it is not possible to free a file from the active pool because it may still be required for a restart, secondary files are formatted and added to the active log file pool.

If the number of secondary files available is used up, requests for most further operations requiring log activity will be refused with an **MQRC_RESOURCE_PROBLEM** being returned to the application.

You use linear logging if you want both restart recovery and media or forward recovery (recreating lost or damaged data by replaying the contents of the log).

Both types of logging can cope with unexpected loss of power assuming that there is no hardware failure.

Checkpointing--ensuring complete recovery

Persistent updates to message queues happen in two stages. First, the records representing the update are written to the log, then the queue file is updated. The log files can thus become more up-to-date than the queue files. To ensure that restart processing begins from a point of consistency, MQSeries uses checkpoints. A checkpoint is a point in time when the record described in the log is the same as the record in the queue. The checkpoint itself consists of the series of log records needed to restart the queue manager: for example, the state of all transactions active at the time of the checkpoint.

Checkpoints are generated automatically by MQSeries. They are taken when the queue manager starts, at shutdown, 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.

MQSeries maintains internal pointers to the head and tail of the log. It moves the head pointer to the most recent checkpoint that is consistent with recovering message data.

Checkpoints are used to make recovery more efficient, and to control the reuse of primary and secondary log files.

Figure 23. Checkpointing. For simplicity, only the ends of the log files are shown.
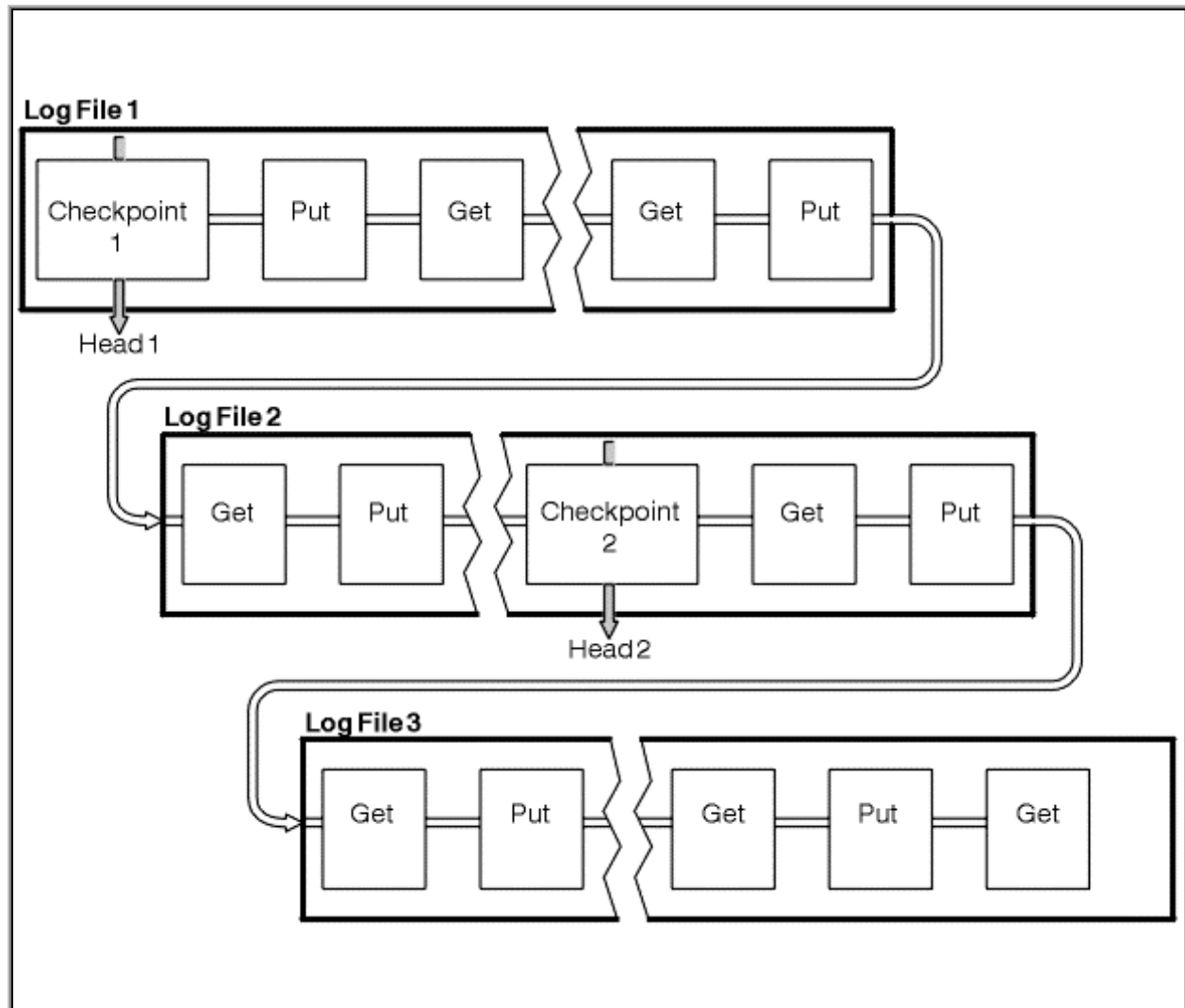
In Figure 23, all records before the latest checkpoint, checkpoint 2, are no longer needed by MQSeries. The queues can be recovered from the checkpoint information and any later log entries. For circular logging, any freed files prior to the checkpoint can be reused. For a linear log, the freed log files no longer need to be accessed for normal operation and become inactive. In the example, the queue head pointer is moved to point at the latest checkpoint, Checkpoint 2, which then becomes the new queue head, head 2. Log File 1 can now be reused.

Figure 24. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.
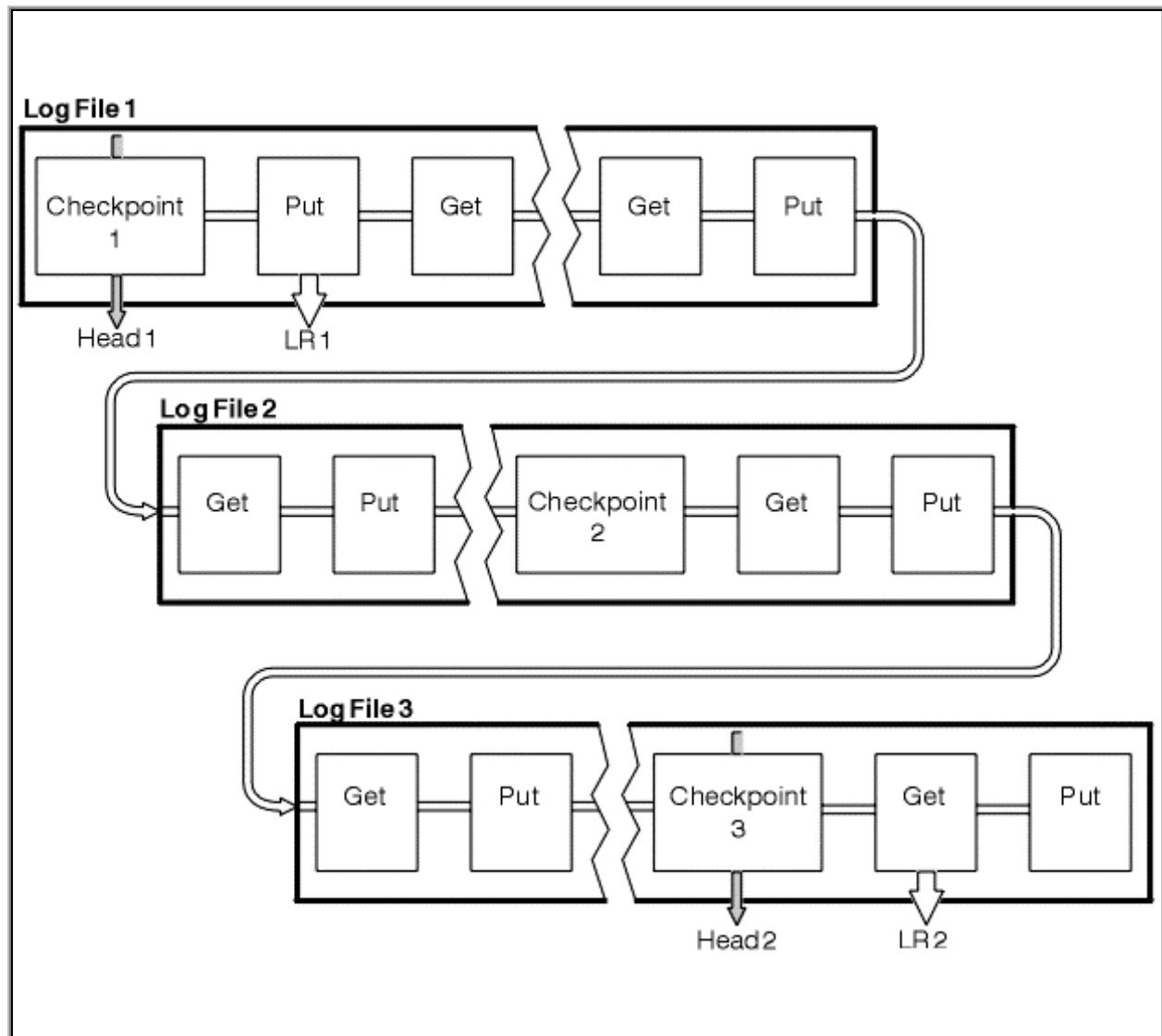
[Figure 24](#) shows how a long running transaction affects reuse of log files. In the example, a long running transaction has caused an entry to the log, shown as LR 1, after the first checkpoint shown. The transaction does not complete, shown as LR 2, until after the third checkpoint. All the log information from LR 1 onwards is retained to allow recovery of that transaction, if necessary, until it has completed.

After the long-running transaction has completed, at LR 2, the head of the log is moved to checkpoint 3, the latest logged checkpoint. The files containing log records prior to checkpoint 3, Head 2, are no longer needed. If you are using circular logging, the space can be reused.

If the primary log files are completely filled before the long-running transaction completes,

secondary log files are used to avoid the risk of a log full situation if possible.

When the log head is moved and you are using circular logging, the primary log files may become eligible for reuse and the logger, after filling the current file, reuses the first primary file available to it. If instead you are using linear logging, the log head is still moved down the active pool and the first file becomes inactive. A new primary file is formatted and added to the bottom of the pool in readiness for future logging activities.

Managing logs

Over time, some of the log records written become unnecessary for restarting the queue manager, and the queue manager reclaims freed space in the log files. This activity is transparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log may fill if a checkpoint has not been taken for a long time, or if a long-running transaction wrote a log record a long time ago. The queue manager tries to take checkpoints sufficiently frequently to avoid the first problem.

When a long-running transaction fills the log, attempts to write log records fail and some MQI calls return **MQRC_RESOURCE_PROBLEM**. (Space is reserved to commit or rollback all in-flight transactions, so **MQCMIT** or **MQBACK** should not fail.)

The queue manager rolls back transactions that consume too much log space. An application whose transaction is rolled back in this way is unable to perform subsequent **MQPUT** or **MQGET** operations specifying syncpoint under the same transaction. An attempt to put or get a message under syncpoint in this state returns **MQRC_BACKED_OUT**. The application may then issue **MQCMIT**, which returns **MQRC_BACKED_OUT**, or **MQBACK** and start a new transaction. When the transaction consuming too much log space has been rolled back, its log space is released and the queue manager continues to operate normally.

If the log fills, a message is issued (**AMQ7463**). In addition, if the log fills because a long-running transaction has prevented the space being released, message **AMQ7465** is issued.

Finally, if records are being written to the log faster than the asynchronous housekeeping processes can handle them, message **AMQ7466** is issued. If you see this message, you should increase the number of log files or reduce the amount of data being processed by the queue manager.

What happens when a disk gets full

The queue manager logging component can cope with a full disk, and with full log files. If the disk containing the log fills, the queue manager issues message **AMQ6708** and an error record is taken.

The log files are created at their maximum size, rather than being extended as log records are written to them. This means that MQSeries can only run out of disk space when it is creating a new file. It therefore cannot run out of space when it is writing a record to the log. MQSeries always knows

how much space is available in the existing log files and manages the space within the files accordingly.

If you fill the drive containing the log files, you may be able to free some disk space. If you are using a linear log, there may be some inactive log files in the log directory which you can copy to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You may be able to reduce the number of primary or secondary log files so that the log does not outgrow the available space. Note that it is not possible to alter the size of the log files for an existing queue manager. The queue manager assumes that all log files are the same size.

Managing log files

If you are using circular logging, ensure that there is sufficient space to hold the log files. You do this when you configure your system (see <u>"Log configuration stanzas"</u>). The amount of disk space used by the log does not increase beyond the configured size, including space for secondary files to be created when required.

If you are using a linear log, the log files are added continually as data is logged, and the amount of disk space used increases with time. If the rate of data being logged is high, disk space is consumed rapidly by new log files.

Over time, the older log files for a linear log are no longer required to restart the queue manager or perform media recovery of any damaged objects. Periodically, the queue manager issues a pair of messages to indicate which of the log files is required:

- Message **AMQ7467** gives the name of the oldest log file needed to restart the queue manager. This log file and all newer log files must be available during queue manager restart.

- Message **AMQ7468** gives the name of the oldest log file needed to do media recovery.

Any log files older than these do not need to be online. You can copy them to an archive medium such as tape for disaster recovery, and remove them from the active log directory. Any log files needed for media recovery but not for restart can also be off-loaded to an archive.

If any log file that is needed cannot be found, operator message **AMQ6767** is issued. Make the log file, and all subsequent log files, available to the queue manager and retry the operation.

Note: When performing media recovery, all the required log files must be available in the log file directory at the same time. Make sure that you take regular media images of any objects you may wish to recover to avoid running out of disk space to hold all the required log files.

Log file location

When choosing a location for your log files, remember that operation is severely impacted if MQSeries fails to format a new log because of lack of disk space.

If you are using a circular log, ensure that there is sufficient space on the drive for at least the configured primary log files. You should also leave space for at least one secondary log file which is needed if the log has to grow.

If you are using a linear log, you should allow considerably more space; the space consumed by the log increases continuously as data is logged.

Ideally, the log files should be placed on a separate disk drive from the queue manager data. This has benefits in terms of performance. It may also be possible to place the log files on multiple disk drives in a mirrored arrangement. This gives protection against failure of the drive containing the log. Without mirroring, you could be forced to go back to the last backup of your MQSeries system.

Using the log for recovery

There are several ways that your data can be damaged. MQSeries for IRIX helps you recover from:

- A damaged data object

- A power loss in the system

- A communications failure

- A damaged log volume

This section looks at how the logs are used to recover from these problems, including:

- ["Recovering from problems"](#)

- ["Recovery scenarios"](#)

Recovering from problems

MQSeries can recover from both communications failures and loss of power. In addition, it is sometimes possible to recover from other types of problem, such as inadvertent deletion of a file.

In the case of a communications failure, messages simply remain on queues until they are removed by a receiving application. If the message is being transmitted, it remains on the transmission queue until it can be successfully transmitted. To recover from a communications failure, it is normally sufficient simply to restart the channels using the link that failed.

If you lose power, when the queue manager is restarted MQSeries restores the queues to their state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when MQSeries stops.

There are ways in which an MQSeries object can become unusable, for example due to inadvertent damage. You then have to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which objects are damaged.

Media recovery

Media recovery means recreating objects from information recorded in a linear log. For example, if an object file is inadvertently deleted, or becomes unusable for some other reason, media recovery

can be used to recreate it. The information in the log required for media recovery of an object is called a media image. Media images may be recorded manually, using the **rcdmqimg** command, or automatically in certain circumstances.

A media image is a sequence of log records containing an image of an object from which the object itself can be recreated.

The first log record required to recreate an object is known as its *media recovery record*; it is the start of the latest media image for the object. The media recovery record of each object is one of the pieces of information recorded during a checkpoint.

When recreating an object from its media image, it is also necessary to replay any log records describing updates performed on the object since the last image was taken.

Consider, for example, a local queue that has an image of the queue object taken before a persistent message is put onto the queue. In order to recreate the latest image of the object, it is necessary to replay the log entries recording the putting of the message to the queue, as well as replaying the image itself.

When an object is created, the log records written contain enough information to completely recreate the object. These records make up the object's first media image. Subsequently, media images are recorded automatically by the queue manager in the following situations:

- Images of all process objects and non-local queues are taken at each shutdown.

- Local queue images are taken when the queue becomes empty.

Media images can also be recorded manually using the **rcdmqimg** command, described in [rcdmqimg (Record media image)](#).

Recovering media images

MQSeries automatically recovers some objects from their media image if it finds that they are corrupt or damaged. In particular, this applies to objects found to be damaged during the normal queue manager startup. If any transaction was incomplete 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 command **rcrmqobj**. This command replays the records in the log to recreate the MQSeries object. The object is recreated from its latest image found in the log, together with all applicable log events between the time the image was saved and the time the recreate command is issued. Should an MQSeries object become damaged, the only valid actions that may be performed are either to delete it or to recreate it by this method. Note, however, that nonpersistent messages cannot be recovered in this way.

See [rcrmqobj (Recreate object)](#) for further details of the **rcrmqobj** command.

It is important to remember that you must have the log file containing the media recovery record, and all subsequent log files, available in the log file directory when attempting media recovery of an object. If a required file cannot be found, operator message **AMQ6767** is issued and the media recovery operation fails. If you do not take regular media images of the objects that you may wish to

recreate, you can get into the situation where you have insufficient disk space to hold all the log files required to recreate an object.

Recovering damaged objects during startup

If the queue manager discovers a damaged object during startup, the action it takes depends on the type of object and whether the queue manager is configured to support media recovery.

If the queue manager object is damaged, the queue manager cannot start unless it can recover the object. If the queue manager is configured with a linear log, and thus supports media recovery, MQSeries automatically tries to recreate the MQSeries object from its media images. If the log method selected does not support media recovery, you can either restore a backup of the queue manager or delete the queue manager.

If any transactions were active when the queue manager stopped, the local queues containing the persistent, uncommitted messages put or got inside these transactions are also needed to start the queue manager successfully. If any of these local queues are found to be damaged, and the queue manager supports media recovery, it automatically attempts to recreate them from their media images. If any of the queues cannot be recovered, MQSeries cannot start.

If any damaged local queues containing uncommitted messages are discovered during startup processing on a queue manager that does not support media recovery, the queues are marked as damaged objects and the uncommitted messages on them are ignored. This is because it is not possible to perform media recovery of damaged objects on such a queue manager and the only action left is to delete them. Message **AMQ7472** is issued to report any damage.

Recovering damaged objects at other times

Media recovery of objects is only automatic during startup. At other times, when object damage is detected, operator message **AMQ7472** is issued and most operations using the object fail. If the queue manager object is damaged at any time after the queue manager has started, the queue manager perform a preemptive shutdown. When an object has been damaged you may delete it or, if the queue manager is using a linear log, attempt to recover it from its media image using the **rcrmqobj** command (see [rcrmqobj (Recreate object)](#) for further details).

Backup and restore

Periodically, you may wish to take a backup of your queue manager data to provide protection against possible corruption due to hardware failures. However, because message data is often short-lived, you may choose not to take backups.

Backing up MQSeries

To take a backup of a queue manager's data, you must first ensure that the queue manager is not running. If your queue manager is running, stop it with the **endmqm** command. If you try to take a backup of a running queue manager, the backup may not be consistent due to updates in progress when the files were copied.

Next, locate the directories under which the queue manager places its data and its log files. You can use the information in the configuration files to determine these directories. For more information about this, refer to [Chapter 13. "Understanding configuration files"](#).

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 you are using MQSeries on. For more information about name transformations, refer to ["Understanding MQSeries file names"](#).

Take copies of all the queue manager's data and log file directories, including all subdirectories. Make sure that you do not miss any of the files, especially the log control file and the configuration files. Some of the directories may be empty, but they will all be required if you restore the backup at a later date, so it is advisable to save them too.

You must also ensure that you preserve the ownerships of the files. You can do this with the tar command.

Restoring MQSeries

If you have to restore a backup of a queue manager, first make sure that the queue manager is not running.

Next, locate the directories under which the queue manager places its data and its log files. This information is held in the configuration file.

Clear out the directories into which you are going to place the backed up data. Finally, copy the backed up queue manager data and log files into the correct places. Check the resulting directory structure to ensure that you have all of the required directories.

Refer to [Appendix C. "Directory structure"](#) for more information about MQSeries directories and subdirectories.

Make sure that you have a log control file as well as the log files. Also check that the MQSeries and queue manager configuration files are consistent so that MQSeries can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager will now start.

Note: Even though the queue manager data and log files are held in different directories, you should back up and restore the directories at the same time. If the queue manager data and log files have different ages, the queue manager is not in a valid state and will probably not start. If it does start, your data will almost certainly be corrupt.

Recovery scenarios

In this section we show a number of possible problems and indicate how to recover from them.

Disk drive failures

You may suffer some problem with a disk drive containing either the queue manager data, the log, or both. Problems can include data loss or corruption. The three cases differ only in the part of the data that survives, if any.

In **all** cases you must first check the directory structure for any damage and, if necessary, repair such damage. If you lose queue manager data, there is a danger that the queue manager directory structure has been damaged. If so, you must recreate the directory tree manually before you try to restart the queue manager. Having checked for structural damage, there are a number of alternative things you can do, depending on the type of logging that you use.

- **Where there is major damage to the directory structure or any damage to the log**, remove all the old files back to the QMgrName level, including the configuration files, the log, and the queue manager directory, restore the last backup and try to restart the queue manager.

- **For linear logging with media recovery**, ensure the directory structure is intact and try to restart the queue manager. If it won't restart, restore a backup. If it restarts, check whether any other objects have been damaged using MQSC. Recover the ones you find, using the **rcrmqobj** command, for example:

```
rcrmqobj -m QMgrName -t * *
```

Where QMgrName is the queue manager being recovered. -t * * indicates that any object of any type will be recovered. If only one or two objects have been reported as damaged, you may want to specify those objects by name and type here.

Note: These commands do not apply to channels.

- **For linear logging with media recovery and with an undamaged log**, you may be able to restore a backup of the queue manager data leaving the existing log files and log control file unchanged. Starting the queue manager applies the changes from the log to bring the queue manager back to its state when the failure occurred.

  This method relies on two facts. Firstly, it is vital that the checkpoint file be restored as part of the queue manager data. This file contains the information determining how much of the data in the log must be applied to give a consistent queue manager.

  Secondly, you must have the oldest log file which was required to start the queue manager at the time of the backup, and all subsequent log files, available in the log file directory.

  If this is not possible, you must restore a backup of both the queue manager data and the log, both of which were taken at the same time.

- **For circular logging, or linear logging without media recovery**, you must restore the queue manager from the latest backup that you have. Once you have restored the backup, restart the queue manager and check as above for damaged objects. However, because you do not have media recovery, you must find other ways of recreating the damaged objects.

Damaged queue manager object

If the queue manager object has been reported as damaged during normal operation, the queue manager performs a pre-emptive shutdown. There are two ways of recovering in these circumstances depending on the type of logging you use:

1. **For linear logging only**, manually delete the file containing the damaged object and restart the queue manager. Media recovery of the damaged object is automatic.

2. **For circular or linear logging**, restore the last backup of the queue manager data and log and restart the queue manager.

Damaged single object

If a single object is reported as damaged during normal operation, there are two ways of recovering, depending on the type of logging you use.

1. **For linear logging**, recreate the object from its media image.

2. **For circular logging**, restore the last backup of the queue manager data and log and restart the queue manager.

Automatic media recovery failure

If a local queue required for queue manager startup with a linear log is damaged, and the automatic media recovery fails, restore the last backup of the queue manager data and log and restart the queue manager.

Chapter 13. Understanding configuration files

MQSeries for IRIX uses *configuration files* to hold basic product configuration information. This chapter describes what they are and how you can use them to change the way that queue managers operate.

What configuration files are

Configuration files define optional values for individual queue managers and for MQSeries on the node as a whole. These files have file name extensions of `ini` and are also referred to as ini files or stanza files.

A configuration file contains one or more *stanzas*, where a stanza is simply a group of lines in the file that together have a common function or define part of a system. For example, there are stanzas associated with logs, with channels, and installable services.

Configuration files may be modified automatically by commands that change the configuration of queue managers on the node and also by editing them manually.

There are two types of configuration file:

- The *MQSeries configuration file*, which specifies values for the MQSeries on the node as a whole. There is one MQSeries configuration file per node.

- *Queue manager configuration files*, which specify values for specific queue managers. There is one queue manager configuration file for each queue manager on the node.

MQSeries configuration file

The MQSeries configuration file `mqs.ini` contains information relevant to all the queue managers on a node. It is created automatically during installation. In particular, the MQSeries configuration file is used to locate the data associated with each queue manager. The MQSeries configuration file is located in the mqm directory, by default `/var/mqm`.

What the MQSeries configuration file contains

The `mqs.ini` file contains the names of the queue managers, the name of the default queue manager, and the location of the files associated with each of them. The following stanzas can appear in `mqs.ini`:

AllQueueManagers

Specifies the path to the qmgrs directory where the files associated with a queue manager are stored. If during installation, you specify a location for MQSeries files that is not the default, the *DefaultPrefix* is automatically changed to reflect this.

DefaultQueueManager

Specifies the default queue manager for the node. This queue manager processes any commands where a queue manager name is not explicitly specified. The stanza is automatically updated if you create a new default queue manager. If you inadvertently create a default queue manager and then wish to revert to the original, you must alter this stanza manually.

QueueManager

There is one such stanza for each queue manager. This specifies the queue manager name and the location of the files associated with that queue manager. The names of these files are based on the queue manager name but these are transformed if the queue manager name is not a valid filename. See "Understanding MQSeries file names".

LogDefaults

Specifies the default log parameters for the node. The *DefaultPrefix* and *DefaultPath* entries allow for the queue manager and its log to be on different physical drives. This is recommended, although by default, they are on the same drive. See "Configuring the logs" for more information about the log file stanzas.

Figure 25 shows an example of an MQSeries configuration file.

Figure 25. Example MQSeries configuration 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=/var/mqm

LogDefaults:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=17
   LogDefaultPath=/var/mqm/log
```

```
QueueManager:
   Name=saturn.queue.manager
   Prefix=/var/mqm
   Directory=saturn!queue!manager

QueueManager:
   Name=pluto.queue.manager
   Prefix=/var/mqm
   Directory=pluto!queue!manager

DefaultQueueManager:
   Name=saturn.queue.manager
```

In this example, MQSeries on the node is using the default locations for queue managers and for the logs.

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.

CAUTION Because the MQSeries configuration file is used to locate the data associated with queue managers, a non-existent 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.

Queue manager configuration file

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. It is created automatically when the queue manager with which it is associated is created.

The file is called `qm.ini` and is held in the root of the directory tree occupied by the queue manager. For example, the path and name for a configuration file for a queue manager called QMNAME is:

`/var/mqm/qmgrs/QMNAME/qm.ini`

Note: The queue manager name can be up to 48 characters in length. However, this does not guarantee that the name is valid or unique, therefore, a directory name is generated based on the queue manager name. This process is known as name transformation, for a description, see ["Understanding MQSeries file names"](#).

What the queue manager configuration file contains

The stanzas that may appear in a queue manager configuration file, `qm.ini` are as follows:

Service

   Specifies the name of one of the installable services, and the number of entry points to that

service. There is one stanza for each service. These services are available:

- Authorization service

- Name service

The authorization service stanza and its associated *ServiceComponent* stanza are added automatically when the queue manager is created.

Once the Object Authority Manager (OAM) has been enabled, you can *only* disable it by:

1. Deleting the queue manager (using the **dltmqm** command)

2. Creating the queue manager again (using the **crtmqm** command) with the MQSNOAUT variable.

The name service stanza must be added manually, if you wish to enable the supplied name service.

ServiceComponent

These stanzas define the service component associated with a particular service. There can be more than one service component stanza for each service, but each service component stanza must match the corresponding service stanza. See *MQSeries Programmable System Management* for more information. By default, the authorization service stanza is present and the associated component, the OAM is active.

Log

Specifies the default log parameters for this queue manager. The fields in this stanza are same as those in the LogDefaults stanza in the mqs.ini file. The values can be made different, if required. See ["Configuring the logs"](#) for more information about the log file stanzas.

Channels

This stanza contains information about the channels. As well as defining the maximum number of channels that can be defined for the queue manager, a second parameter limits the number of channels that can be active at any time.

The MaxChannels number is the maximum number of channels that can have current status. This includes channels which are retrying or stopped.

The MaxActiveChannels parameter limits the number of channels that can have current status at the same time.

See the *MQSeries Distributed Queuing Guide* for more information about channels.

TCP

Specifies network protocol configuration parameters. These stanzas override the default parameters for channels. Only stanzas representing changed default values are actually present.

KeepAlive, if specified, causes TCP/IP to periodically check that the other end of the

connection is still available. If it is not, the channel is closed.

You can set the TCP/IP keepalive parameters by using the `idtune` and `idbuild` commands to modify the TCP_KEEPCNT and TCP_KEEPINT values for the kernel configuration. The default configuration is to retry 7 times at 7200 second (2 hourly) intervals.

See the *MQSeries Distributed Queuing Guide* for more information.

Figure 26 shows how the stanzas might be arranged in a queue manager configuration file.

Figure 26. Example 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    *#
#*                                                                *#
#******************************************************************#
Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=mqmtop/bin/amqzfu.o
   ComponentDataSize=0

Service:
   Name=NameService
   EntryPoints=9

ServiceComponent:
   Service=NameService
   Name=MQSeries.UNIX.name.service
   Module=/u/opman/abctest.o
   ComponentDataSize=128

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=17
   LogPath=/var/mqm/log/saturn!queue!manager/

CHANNELS:
  MaxChannels = 20          ; Maximum number of Channels allowed,
                            ; the default number 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
  Port = 1800               ; use port 1800 instead of the default 1414
  KeepAlive = Yes           ; Switch KeepAlive on

```

Editing configuration files

You can edit the default configuration files to alter the system defaults. However, before editing any configuration file, make sure that you have a backup that you can revert to.

In some circumstances, you may have to edit your configuration files. For example:

- If you lose a configuration file; recover from backup if possible.

- If you need to move one or more queue managers to a new directory.

- If you need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.

- When advised to do so by your Willow Technology Support Center.

Changing the default prefix

If you change the default prefix, DefaultPrefix, for the message queue manager, you must replicate the directory structure that was created at installation time (see [Figure 34](#)). In particular, the qmgrs structure must be created. You must stop MQSeries before changing the default prefix. Only restart MQSeries after the structures have been moved to the new location and the default prefix has been changed.

Implementing changes to configuration files

If you edit a configuration file the changes will not be implemented immediately by the queue manager. Changes made to the MQSeries configuration file will only be implemented when MQSeries is started. Changes made to a queue manager configuration file will be implemented when the queue manager is started. If the queue manager is running when you make the changes, you must stop and then restart the queue manager for any changes to be recognized by the system.

Recommendations for configuration files

When you create a new queue manager, you should:

- Back up the MQSeries configuration file

- Back up the new queue manager configuration file

Configuring the logs

The log parameters in the MQSeries configuration file are used as default values when you create a queue manager. These defaults can be overridden if you specify the log parameters on the **crtmqm** command. See [crtmqm (Create queue manager)](#) for details of this command.

The values specified in the queue manager configuration file are read when the queue manager is started. The file is created when the queue manager is created.

The values in a configuration file are set according to these priorities:

1. Parameters entered on the command line override both the queue manager configuration file and the MQSeries configuration file.

2. The queue manager configuration file overrides the MQSeries configuration file.

3. The MQSeries configuration file contains the supplied default values.

Note: User ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the logs files are in the default locations supplied with the product.

If you use a value that is not valid in a configuration file, it is ignored. The effect is the same as missing out the value entirely. An operator message is issued to indicate the problem.

You can edit the MQSeries configuration file after installation and change the default values to your own requirements.

Log configuration stanzas

The size and location of the log is configured by stanzas in the MQSeries and queue manager configuration files. These stanzas specify the type of logging to be used, the log file size, and the log path.

The MQSeries configuration file contains a stanza called *LogDefaults* with the following format:

```
LogDefaults:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=17
   LogDefaultPath=/var/mqm/log
```

The values specified in the MQSeries configuration file are read whenever a queue manager is created, started, or deleted.

Each queue manager configuration file has a stanza called *Log*, which has the following format:

```
Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=17
   LogPath=/var/mqm/log/<QM_Dir_Name>/
```

<QM_Dir_Name> is the sub-directory name for this queue manager, providing a unique path to the logs. This is the queue manager name if it is a valid for the file system; otherwise, it is a transformed name. See ["Understanding MQSeries file names"](#).

LogPrimaryFiles

Primary log files are the log files allocated during creation for future use.

The default number is 3.

The default can be overridden by editing the *LogPrimaryFiles* value in the product and queue manager configuration files.

The value is examined when the queue manager is created or started. You can increase or decrease it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted and the effect may not be immediate.

The minimum number of primary log files is 2 and the maximum is 62. The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogSecondaryFiles

Secondary log files are the log files allocated when the primary files are exhausted.

The default number is 2.

The default can be overridden using the *LogSecondaryFiles* value in the product and queue manager configuration files.

The value is examined when the queue manager is created or started. You can change this value, but changes are not effective until the queue manager is restarted and the effect may not be immediate.

The minimum number of secondary log files is 1 and the maximum is 61. The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogFilePages

The log data is held in a series of files called log files.

The default number of log file pages is 1024, equating to a log file size of 4 MB. The minimum number of log file pages 64 and the maximum is 16 384.

The log file size is specified in units of 4 KB pages. It can be specified only during queue manager creation and the value used is obtained by taking the default (1024) and overriding it with the value in the *LogFilePages* attribute in the MQSeries configuration file, or by overriding with the value specified on the **crtmqm** command using the `-lf` flag.

Note: The size of the log files is specified during queue manager creation and cannot be changed for an existing queue manager.

LogType

The *LogType* parameter is used to define the type to be used, either CIRCULAR or LINEAR.

The default is CIRCULAR.

If you wish to change the default, you may either edit the MQSeries configuration file or specify linear logging with the **crtmqm** command. You cannot change the logging method after a queue manager has been created.

LogBufferPages

The amount of memory allocated to buffer records for writing is configurable. The size of the buffers is specified in units of 4 KB pages.

The default number of buffer pages is 17, equating to 68 KB.

The default can be overridden using the *LogBufferPages* value in the MQSeries and queue manager configuration files.

The value is examined when the queue manager is created or started and may be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

The minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

LogPath

You can specify the directory in which the log files for a queue manager reside. The directory should exist on a local device to which the queue manager can write and, preferably, should be on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is `/var/mqm/log`.

You can specify the name of a directory in the **crtmqm** command using the `-ld` flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the Log File Path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify `-ld` on the **crtmqm** command, the value of the `LogDefaultPath` attribute in the MQSeries configuration file is used. If this attribute is missing, the default of

`/var/mqm/log` is used. The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager has been created, a `LogPath` value is created in the log stanza in the queue manager configuration file giving the complete directory name for the queue manager's log. This value is used to locate the log when the queue manager is started or deleted.

Specifying log file sizes

The size of the log file that you require depends on the number and size of messages that are to be handled by your system. Each operation adds an overhead to the size of the log. For example, when a persistent message is put to a queue, the message data must be written to the log to make recovery of the message possible. The message descriptor is also logged together with some internal information that describes the effect of putting the message on the queue.

There is a trade-off between the size of your log files and the number of files that you have. Larger files are more difficult to handle but are more efficient.

Figure 27 shows approximate values for the header information required for various types of operation.

Figure 27. Log overhead sizes

(All values are approximate.)	
Operation	**Size**
Put persistent message	600 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, roll-back	1000 bytes + 12 bytes for each get or put to be rolled back
Create object	1500 bytes
Delete object	300 bytes
Alter attributes	1024 bytes
Record media image	800 bytes + image The image is divided into segments of 15700 bytes, each having a 300-byte overhead.

Checkpoint	750 bytes + 200 bytes for each active unit of work. Additional data may be logged for any uncommitted puts or gets that have been buffered for performance reasons.

Chapter 14. Problem Determination

This chapter suggests reasons for any transient problems you may have using MQSeries for IRIX. The process of problem determination is that you start with the symptoms and trace them back to their cause. Not all problems can be solved immediately, for example, performance problems caused by the limitations of your hardware. Also, if you think that the cause of the problem is in the MQSeries code, contact your Willow Technology Support Center. This chapter contains these sections:

- ["Preliminary checks"](#)
- ["Common programming errors"](#)
- ["Understanding MQSeries file names"](#)
- ["What to do next"](#)
- ["Application design considerations"](#)
- ["Incorrect output"](#)
- ["Error logs"](#)
- ["Dead-letter queues"](#)
- ["Configuration files and problem determination"](#)
- ["Using trace"](#)
- ["First failure support technology"](#)
- ["Problem determination with clients"](#)

Preliminary checks

The cause of your problem could be in:

- MQSeries
- The network
- The application

The sections that follow raise some fundamental questions that you need to consider. Work through the questions, making a note of anything that might be relevant to the problem.

Has MQSeries for IRIX run successfully before?

If MQSeries has not run successfully before, it is likely that you have not yet set it up correctly. See [Chapter 2. "Installing MQSeries for IRIX"](#) to check that you have carried out all the steps correctly.

Are there any error messages?

MQSeries uses 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. Check the error logs to see if any messages have been recorded that are associated with your problem.

See ["Error logs"](#) for information about the contents of the error logs and their locations.

Are there any return codes explaining the problem?

If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, refer to the *MQSeries Application Programming Reference* manual for a description of that return code.

Can you reproduce the problem?

If you reproduce the problem, consider the conditions under which it can be reproduced:

- Is it caused by a command or an equivalent administration request?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.

- Is it caused by a program? Does it fail on all MQSeries systems and all queue managers, or only on some?

- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.

Have any changes been made since the last successful run?

When you are considering changes that might recently have been made, think about the MQSeries system, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you are not aware of might have been run on the system.

- Have you changed, added, or deleted any queue definitions?

- Have you changed or added any channel definitions? Changes may have been made to either MQSeries channel definitions or any underlying communications definitions required by your application.

- Do your applications deal with return codes that they might get as a result of any changes you have made?

Has the application run successfully before?

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Before you answer **Yes** to this question, consider the following:

- Have any changes been made to the application since it last ran successfully?

  If so, it is likely that the error lies somewhere in the new or modified part of the application. Take a look at the changes and see if you can find an obvious reason for the problem. Is it possible to retry using a backlevel of the application?

- Have all the functions of the application been fully exercised before?

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

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

- Does the application check all return codes?

  Has your MQSeries system been changed, perhaps in a minor way, but your application does not check the return codes it receives as a result of the change. For example; does your application assume that the queues it accesses 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?

- Does the application run on other MQSeries systems?

  Could it be that there is something different about the way that this MQSeries system is set up which is causing the problem? For example, have the queues been defined with the same message length or priority?

If the application has not run successfully before

If your application has not yet run successfully, you need to examine it carefully to see if you can find any errors.

Before you look at the code, and depending upon which programming language the code is written in, examine the output from the translator, or the compiler and linkage editor, if applicable, to see if any errors have been reported.

If your application fails to translate, compile, or link-edit into the load library, it will also fail to run if you attempt to invoke it. See the *MQSeries Application Programming Reference* manual for information about building your application.

If the documentation shows that each of these steps was accomplished without error, you should consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See "Common programming errors" for some examples of common errors that cause problems with MQSeries applications.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of MQSeries has been started.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes that might account for the problem or changed any MQSeries definitions?

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, so these are the times when load-dependent problems are most likely to occur. (If your MQSeries network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Is the problem intermittent?

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

Have you applied any service updates?

If a service update has been applied to MQSeries, check that the update action completed successfully and that no error message was produced.

- Did the update have any special instructions?

- Was any test run to verify that the update had been applied correctly and completely?

- Does the problem still exist if MQSeries is restored to the previous service level?

- If the installation was successful, check with the Willow Technology Support Center for any patch error.

- If a patch has been applied to any other program, consider the effect it might have on the way MQSeries interfaces with it.

Common programming errors

The examples that follow illustrate the most common causes of problems encountered while running MQSeries programs. You should consider the possibility that the problem with your MQSeries system could be caused by one or more of these errors:

- Assuming that queues can be shared, when they are in fact exclusive.

- Passing incorrect parameters in an MQI call.

- Passing insufficient parameters in an MQI call. This may mean that MQI cannot set up completion and reason codes for your application to process.

- Failing to check return codes from MQI requests.

- Passing variables with incorrect lengths specified.

- Passing parameters in the wrong order.

- Failing to initialize *MsgId* and *CorrelId* correctly.

Problems with commands

You should be careful when including special characters in descriptive text for some commands. For example, back slash, \, and double quote, ", characters. If you use either of these in descriptive text, precede them with a \, that is, enter \\ or \" if you want \ or " in your text.

Understanding MQSeries file names

The path to a queue manager directory is formed from the following:

- A prefix - the first part of the name:

  `/var/mqm`

  This prefix is defined in the queue manager configuration file.

- 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`

  would be represented as:

  `queue!manager`

This process is referred to as **name transformation**.

Queue manager name transformation

In MQSeries you can give a queue manager a name containing up to 48 characters. For example, you could name a queue manager:

`QUEUE.MANAGER.ACCOUNTING.SERVICES`

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

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

1. Transform individual characters:

   . becomes !

   / becomes &

2. If the name is still not valid:

   1. Truncate it to eight characters

   2. Append a three character numeric suffix

For example, assuming the default prefix, the queue manager name becomes:

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

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

Object name transformation

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

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

What to do next

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 (see "MQSeries publications") and in the libraries of other licensed programs.

If you have not yet found the cause, you must start to look at the problem in greater detail.

The purpose of this section is to help you identify the cause of your problem if the preliminary checks have not enabled you to find it.

When you have established that no changes have been made to your system, and that there are no problems with your application programs, choose the option that best describes the symptoms of your problem.

- "Have you obtained some incorrect output?"

- "Have you failed to receive a response from a PCF command?"

- "Does the problem affect only remote queues?"

- "Is your application or operating system running slowly?"

If none of these symptoms describe your problem, consider whether it might have been caused by another component of your system.

Have you obtained some incorrect output?

In this book, "incorrect output" refers to your application:

- Not receiving a message that it was expecting.

- Receiving a message containing unexpected or corrupted information.

- Receiving a message that it was not expecting, for example, one that was destined for a different application.

In all cases, check that any queue or queue manager aliases that your applications are using are correctly specified and accommodate any changes that have been made to your network.

If an MQSeries error message is generated, all of which are prefixed with the letters AMQ, you should look in the error log. See "Error logs" for further information.

Have you failed to receive a response from a PCF 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 **dspmqcsv** command to check the status of the command server.

   - If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it.

   - If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is

not enabled for **MQGET** requests, enable the queue for **MQGET** requests.

- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *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 browse sample application (AMQSBCG) provided 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"](#) for further information.

- Are the queues enabled for put and get operations?

- Is the *WaitInterval* set to a sufficient length?

  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?

  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 show 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 Willow Technology Support Center for help.

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems:

1. Display the information about each queue. You can use the MQS command DISPLAY

QUEUE to display the information.

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 great? That is, not causing a trigger event to be generated often enough?

    Is the process name correct?

    Is the process available and operational?

  - Can the queue be shared? If not, another application could already have it open for input.

  - Is the queue enabled appropriately for GET and PUT?

- If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the **MQOPEN** call has failed for some reason.

  Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. Note that the values may have changed and that the queue was open but is now closed.

  You need to check the status at the time you expect to put or get a message.

If you are unable to solve the problem, contact your Willow Technology Support Center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues, check the following:

- Check that required channels have been started, can be triggered, and that any required initiators are running.

- Check that the programs that should be putting messages to the remote queues have not reported problems.

- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.

- Check the error logs for messages indicating channel errors or problems.

- If necessary, start the channel manually. See the *MQSeries Distributed Queuing Guide* for information about how to do this.

See the *MQSeries Distributed Queuing Guide* for information about how to define channels.

Is your application or operating system 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.)

A performance problem may be caused by a limitation of your hardware.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that MQSeries is running slowly:

- Your system is slow to respond to MQSeries commands.

- Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem may lie with MQSeries for IRIX itself. If you suspect this, you need to contact your Willow Technology Support Center for assistance.

Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making MQSeries 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 allows messages to hold up to 4MB 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 logged. Logging 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. Using the **MQGET** call in this way affects the performance of your application.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the **MQGET** call with the *BufferLength* field set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second **MQGET** call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the **MQINQ** call to find the maximum size of messages that the queue can accept, then use this value in your **MQGET** call. The maximum size of messages for a queue is stored in the *MaxMsgLength* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be as high as 4MB, the maximum allowed by MQSeries for IRIX.

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 inaccessible, while other tasks might be waiting to get these messages. This has implications in terms of storage, and in terms of threads tied up with tasks that are attempting to get messages.

Use of the MQPUT1 call

Use the **MQPUT1** call only if you have a single message to put on a queue. If you want to put more than one message, use the **MQOPEN** call, followed by a series of **MQPUT** calls and a single **MQCLOSE** call.

Incorrect output

The term "incorrect output" can be interpreted in a lot of different ways. The meaning for the purpose of problem determination within this book is explained in <u>"Have you obtained some incorrect output?"</u>.

Two types of incorrect output are discussed in this section:

- Messages that do not appear when you are expecting them
- Messages that contain the wrong information, or information that has been corrupted

Additional problems that you might find if your application includes the use of distributed queues are also discussed.

Messages do not appear on the queue

If messages do not appear when you are expecting them, check for the following:

- Has the message been put on the queue successfully?
    - Has the queue been defined correctly, for example is MAXMSGL sufficiently large?
    - Is the queue enabled for putting?
    - Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Are you able to get any messages from the queue?
    - Do you need to take a syncpoint?
    - If messages are being put or retrieved within syncpoint, they are not available to other tasks until the unit of recovery has been committed.
    - Is your wait interval long enough?
    - You can set the wait interval as an option for the **MQGET**. You should ensure that you are waiting long enough for a response.
    - Are you waiting for a specific message that is identified by a message or correlation identifier (*MsgId* or *CorrelId*)?
    - Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful **MQGET** call sets both these values to that of the message retrieved, so you 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 has been restarted, the message has been lost.
    - Has another application got exclusive access to the queue?

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

- Did the application get started?

  If it should have been triggered, check that the correct trigger options were specified.

- Did the application stop?

- Is a trigger monitor running?

- Was the trigger process defined correctly?

- Did the application complete correctly?

  Look for evidence of an abnormal end in the job log.

- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an **MQGET** call with a buffer length of zero to find out the length of the message, and then issues a specific **MQGET** call specifying the *MsgId* of that message. However, in the meantime, another transaction issues a successful **MQGET** call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a 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 onto the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, the format of the message data may have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.

- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not really intended for an application servicing a different queue. If 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?

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

Problems with incorrect output when using distributed queues

If your application uses distributed queues, you should also consider the following points:

- Has MQSeries been correctly installed on both the sending and receiving systems, and correctly configured for distributed queuing?

- Are the links available between the two systems?

  Check that both systems are available, and running to MQSeries. Check that the connection between the two systems and the channels between the two queue managers are active.

- Is triggering set on in the sending system?

- Is the message you are waiting for a reply message from a remote system?

  Check that triggering is activated in the remote system.

- Is the queue already full?

  This could mean that an application was unable to put the required message onto the queue. If this is so, check if the message has been put onto the dead-letter queue.

  The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure.

- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.

- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap stops the distributed queuing component. See the *MQSeries Distributed Queuing Guide* for more information about distributed queuing.

- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic conversion occurs when the **MQGET** is issued if the format is recognized as one of the built-in formats.

  If the data set is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

  An exception to the above occurs if you are sending data to MQSeries for MVS/ESA. Here the conversion occurs on message transmission from MQSeries for OS/2.

Refer to the *MQSeries Distributed Queuing Guide* for further details of data conversion.

Error logs

MQSeries for IRIX 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 the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client.

1. If the queue manager name is known and the queue manager is available:

   `/var/mqm/qmgrs/QMgrName/errors/AMQERR01.LOG`

2. If the queue manager is not available:

   `/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG`

3. If an error has occurred with a client application:

   `AMQERR01.LOG`

   in the path for the environment.

4. First Failure Support Technology (FFST) - see ["How to examine the FFSTs"](#).

Note: In the case of clients, the errors are stored on the client's root drive.

Log files

At installation time an @SYSTEM/errors directory is created in the QMGRS file path. The errors subdirectory can contain up to three error log files named:

- AMQERR01.LOG

- AMQERR02.LOG

- AMQERR03.LOG

After you have created a queue manager, 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 256KB. The files are placed in the errors subdirectory of each queue manager that you create.

As error messages are generated they are placed in AMQERR01. When AMQERR01 gets bigger than 256KB 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 qmgr/@SYSTEM/errors subdirectory.

To examine the contents of any error log file, use your usual editor.

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 on the root directory, mqm.

If the MQSeries configuration file is readable, and the DefaultPrefix attribute of the AllQueueManagers stanza is readable, errors are logged in the DefaultPrefix/errors directory.

For further information about configuration files, see [Chapter 13. "Understanding configuration files"](#).

Operator messages

In MQSeries for IRIX, 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 (NLS) enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any, and are also written to the error log AMQERR01.LOG in the queue manager directory. For example:

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

Some errors are logged to the AMQERR01.LOG file in the queue manager directory and others to the @SYSTEM directory copy of the error log.

Example error log

This example shows part of a section of an MQSeries for IRIX error log:

```
 ...
08/01/95  11:41:56 AMQ8003: MQSeries queue manager started.
EXPLANATION: MQSeries queue manager janet started.
ACTION: None.
-------------------------------------------------------------------
08/01/95  11:56:52 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-------------------------------------------------------------------
08/01/95  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.
-------------------------------------------------------------------
08/01/95  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.
-------------------------------------------------------------------
08/01/95  14:28:57 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
-------------------------------------------------------------------
08/02/95  15:02:49 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-------------------------------------------------------------------
08/02/95  15:02:51 AMQ9001: Channel program ended normally.
EXPLANATION: Channel program 'JANET' ended normally.
ACTION: None.
08/02/95  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.
-------------------------------------------------------------------
08/02/95  15:09:32 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
 ...
```

Dead-letter queues

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the browse sample application (AMQSBCG) provided to browse messages on the queue using **MQGET**. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

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

Problems may occur if you do not have a dead-letter queue on each queue manager you are using. The supplied sample program amqscoma.tst creates the default objects for a queue manager, including a dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found and result in **queue manager unavailable** type errors.

There are several checks you can make on the configuration files:

- Ensure that the configuration files exist

- Ensure that they have appropriate permissions, for example:

  ```
  rw-rw-r--   1 mqm       mqm            1371 Sep 17 14:32 /var/mqm/mqs.ini
  ```

- Ensure that the MQSeries configuration file references the correct queue manager and log directories

---

Using trace

MQSeries for IRIX uses the following commands for the trace facility:

- **strmqtrc** - see [strmqtrc (Start MQSeries trace)](#)

- **dspmqtrc** - see [dspmqtrc (Display MQSeries formatted trace output)](#)

- **endmqtrc** - see [endmqtrc (End MQSeries trace)](#)

The trace facility uses one file for each entity being traced, in which trace information is recorded.

Files associated with trace are created in a fixed location in the file tree, which is `/var/mqm/trace`.

All queue managers, all early tracing and all @SYSTEM tracing takes place to files in this directory.

Note: It is possible to accommodate production of large trace files by mounting a temporary filesystem over this directory.

File names for trace files.

Trace file names are constructed in the following way:

```
AMQppppp.TRC
```

where **ppppp** is the PID of the process producing the trace.

Notes:

1. The value of the process id can contain fewer, or more, digits than shown in the example.

2. There will be one trace file for each process running as part of the entity being traced.

Example trace data

The following example is an extract of an MQSeries for IRIX trace:

Figure 28. Sample MQSeries for IRIX trace

```
 ...
ID  ELAPSED_SEC DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

30D 3.290727552 0.000000    MQS FNC Exit.... 12884.1 xllListenSelectAccept..
30D 3.290787328 0.059776    MQS CEI Entry... 12884.1 xcsFreeQuickCell 0
30D 3.290820736 0.033408    MQS CEI Entry.... 12884.1 xllSpinLockRequest 0
30D 3.290841088 0.020352    MQS CEI Entry..... 12884.1 xllCompareAndSwap 0
30D 3.290861440 0.020352    MQS FNC Exit...... 12884.1 xllCompareAndSwap ..
30D 3.290875520 0.014080    MQS FNC Exit..... 12884.1 xllSpinLockRequest ..
30D 3.291131520 0.256000    MQS FNC Entry.... 12884.1 xstFreeCell 0
30D 3.291151744 0.020224    MQS FNC Exit..... 12884.1 xstFreeCell rc=00000000
30D 3.291255296 0.103552    MQS CEI Entry.... 12884.1 xllSpinLockRelease 0
30D 3.291274112 0.018816    MQS FNC Exit..... 12884.1 xllSpinLockRelease ..
30D 3.291292288 0.018176    MQS CEI Exit.... 12884.1 xcsFreeQuickCell rc=..
30D 3.291317632 0.025344    MQS FNC Exit!.. 12884.1 xllWaitSocketEvent
rc=108..
30D 3.291337344 0.019712    MQS CEI Exit!. 12884.1 xcsWaitEventSem rc=10806020
30D 3.291366528 0.029184    MQS CEI Exit! 12884.1 zcpReceiveOnLink rc=20805311
30D 3.291398400 0.031872    MQS FNC Entry 12884.1 zxcProcessChildren 0
30D 3.291752448 0.354048    MQS CEI Entry. 12884.1 xcsRequestMutexSem 0
 ...
```

Notes:

1. In this example the data is truncated. In a real trace, the complete function names and return codes are present.

2. The return codes are given as values, not literals.

First failure support technology

Information which, on the OS/2 and AIX platforms, is normally recorded in FFST logs is, on MQSeries for IRIX, recorded in a file in the `/var/mqm/errors` directory.

These errors are normally severe, unrecoverable errors and indicate either a configuration problem with the system or an MQSeries internal error.

How to examine the FFSTs

The files are named `AMQnnnnn.mm.FDC`, where:

nnnnn is the process id reporting the error

mm is a sequence number, normally 0.

When a process creates an FFST it also sends a record to syslog. The record contains the name of the FFST file to assist in automatic problem tracking.

The syslog entry is made at the "user.error" level. See the MQSeries for IRIX documentation on `syslog.conf` for details on how to configure this.

A typical FFST is shown in Figure 29.

Figure 29. Sample MQSeries for IRIX First Failure Symptom Report.

```
  MQSeries First Failure Symptom Report
  =====================================

  Date/Time          :- Friday July 14 14:06:52 BST 1995
  Host Name          :- unknown
  PIDS               :- 5697-176
  LVLS               :- 220
  Product Long Name  :- MQSeries for IRIX
  Vendor             :- Willow Technology, Inc.
  Probe Id           :- XC130003
  Application Name   :- MQM
  Component          :- xehExcepti
  Build Date         :- Jul 14 1998
  Userid             :- 00000231 (mqm)
  Process            :- 00015967
  Major Errorcode    :- xecSTOP
  Minor Errorcode    :- OK
  Probe Type         :- HALT6109
  Probe Severity     :- 1
  Probe Description  :- AMQ6125: An internal MQSeries error has occurred.
  Arith1             :- 11 b

MQM Function Stack
xllTidyUpSems
xcsFFST

MQM Trace History
 ...
```

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

However, there is one set of problems that they may be able to solve. If the FFST shows "out of resource", "out of space on device", or "invalid parameter" descriptions when calling one of the IPC functions, for example, `semop` or `shmget`, it is likely that the relevant kernel parameter limit has been exceeded.

If the FFST shows a problem with `setitimer`, it is likely that a change to the kernel timer parameters is needed.

To resolve these problems, increase the IPC limits, rebuild the kernel and restart the machine. See

for further details.

Problem determination with clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are now additional reason codes for error conditions associated with clients. For example:

- Remote machine not responding

- Communications line error

- Invalid machine address

The most common time for errors to occur is when an application issues an **MQCONN** and receives the response MQRC_Q_MQR_NOT_AVAILABLE. An error message, written to the client log file, explains the cause of the error. Messages may also be logged at the server depending on the nature of the failure.

Terminating clients

Even though a client has terminated it is still possible for the process at the server to be holding its queues open. Normally, this will only be for a short time until the communications layer detects that the partner has gone.

Error messages with clients

When an error occurs with a client system, error messages are put into the error files associated with the server, if possible. If the error cannot be placed there, the client code attempts to place the error message in an error log in the root directory of the client machine.

OS/2 and UNIX systems clients

Error messages for OS/2, and UNIX systems clients are placed in the error logs in the same way as they are for the respective MQSeries server systems. Typically, these files appear in `/var/mqm/errors`.

For more information on clients see the *MQSeries Clients* book.

DOS and Windows clients

The location of the log file **AMQERR01.LOG** is set by the MQDATA environment variable. The default location, if not overridden by MQDATA, is:

```
C:\
```

Working in the DOS environment involves the environment variable MQDATA.

This is the default library used by the client code to store trace and error information; it also holds the directory name that the **qm.ini** file is stored in (needed for NetBIOS setup). If not specified, it

defaults to the C drive.

The names of the default files held in this library are:

AMQERR01.LOG

For error messages.

AMQERR01.FDC

For First Failure Data Capture messages.

Part 2. Reference section

Chapter 15. MQSeries control commands

This chapter contains reference material for the control commands used with MQSeries for IRIX. All commands in this chapter can be issued from an IRIX shell. These commands are case-sensitive.

Names

In general, the names of MQSeries objects can have up to 48 characters. This rule applies to all the following objects:

- Queue managers
- Queues
- Process definitions

The maximum length of channel names is 20 characters.

The characters that can be used for all MQSeries names are:

- Uppercase A-Z
- Lowercase a-z
- Numerics 0-9
- Period (.)
- Underscore (_)
- Forward slash (/) (see note *(NMSNT)*)
- Percent sign (%) (see note *(NMSNT)*)

Notes:

1. *(NMSNT)* Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks everywhere it is used.

2. Leading or embedded blanks are not allowed.

3. National language characters are not allowed.

4. Names may be enclosed in double quotation marks, but this is only essential if special characters are included in the name.
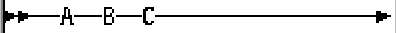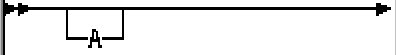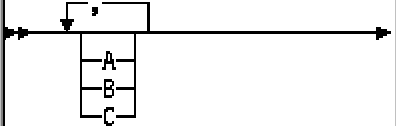
How to read syntax diagrams

This chapter contains syntax diagrams (sometimes referred to as "railroad" diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

Figure 30. How to read syntax diagrams

Convention	Meaning
►►──A──B──C────────────►	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
►►────────────────────► └─A─┘	You may specify value A. Optional values are shown below the main line of a syntax diagram.
►►──┬─A─┬──────────────► ├─B─┤ └─C─┘	Values A, B, and C are alternatives, one of which you must specify.
►►────────────────────► ├─A─┤ ├─B─┤ └─C─┘	Values A, B, and C are alternatives, one of which you may specify.
►►──┬─────┬────────────► ├─A─┤ ├─B─┤ └─C─┘	You may specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	Values A, B, and C are alternatives, one of which you may specify. If

	you specify none of the values shown, the default A (the value shown above the main line) is used.
	The syntax fragment `Name` is shown separately from the main syntax diagram.

Example syntax diagram

Here is an example syntax diagram that describes the **hello** command:
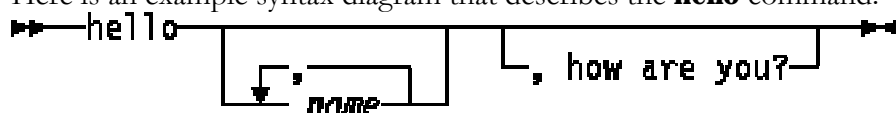
According to the syntax diagram, these are all valid versions of the **hello** command:

```
hello
hello name
hello name, name
hello name, name, name
hello, how are you?
hello name, how are you?
hello name, name, how are you?
hello name, name, name, how are you?
```

Note that the space before the *name* value is significant, and that if you do not code *name* at all, you must still code the comma before `how are you?`.

Syntax help

You can obtain help about the syntax of any of the commands in this chapter by entering the command followed by a question mark. MQSeries responds by listing the syntax required for the selected command. The syntax shows all the parameters and variables associated with the command. Various parentheses are used to indicate whether an item in the list is required or not:

```
CmdName [-x OptParam ] ( -c | -b ) { -p principal } argument
```

Where

CmdName

The command name for which help has been requested

[-x OptParam]

The brackets indicate that this is an optional parameter.

(-c | -b)

A mandatory field. In the case shown, you must select one of the flags c or b

{ -p principal }

An optional list of variables that you may supply, but, if this is shown, at least one variable must be provided when you enter the command.

argument

An argument required to be supplied with this command, mandatory if shown on the response to the query.

Examples

1. Result of entering `endmqm ?`

   ```
   endmqm [-z][-c | -i | -p] QMgrName
   ```

2. Result of entering `rcdmqimg ?`

   ```
   rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]
   ```

crtmqcvx (Data conversion)

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert your C structures.

The command reads an input file containing a structure or structures to be converted. It then writes an output file containing a code fragment or fragments to convert those structures.

For further information about this command and how to use it, refer to the *MQSeries Distributed Queuing Guide.*

```
►►──crtmqcvx──SourceFile──TargetFile──────────────────────────────►◄
```

Flags/parameters

SourceFile

Specifies the input file containing the C structures to be converted.

TargetFile

Specifies the output file containing the code fragments generated to convert the structures.

Return codes

0   Command completed normally

10   Command completed with unexpected results

20   An error occurred during processing

Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source.tmp target.c
```

The input file, `source.tmp` looks like this:

```
/* This is a test C structure which can be converted by the */
/* crtmqcvx utility                                          */

struct my_structure
{
```

```
    int    code;
    MQLONG value;
};
```

The output file, target.c, produced by the command is shown below. You can use these code
fragments in your applications to convert data structures. However, if you do so, you should
understand that the fragment uses macros supplied in the MQSeries header file amqsvmha.h.

```
MQLONG Convertmy_structure(
            PMQBYTE *in_cursor,
            PMQBYTE *out_cursor,
            PMQBYTE in_lastbyte,
            PMQBYTE out_lastbyte,
            MQHCONN hConn,
            MQLONG  opts,
            MQLONG  MsgEncoding,
            MQLONG  ReqEncoding,
            MQLONG  MsgCCSID,
            MQLONG  ReqCCSID,
            MQLONG  CompCode,
            MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}
```

crtmqm (Create queue manager)

Purpose

Use the **crtmqm** command to create a local queue manager. Once a queue manager has been created, use the **strmqm** command to start it.
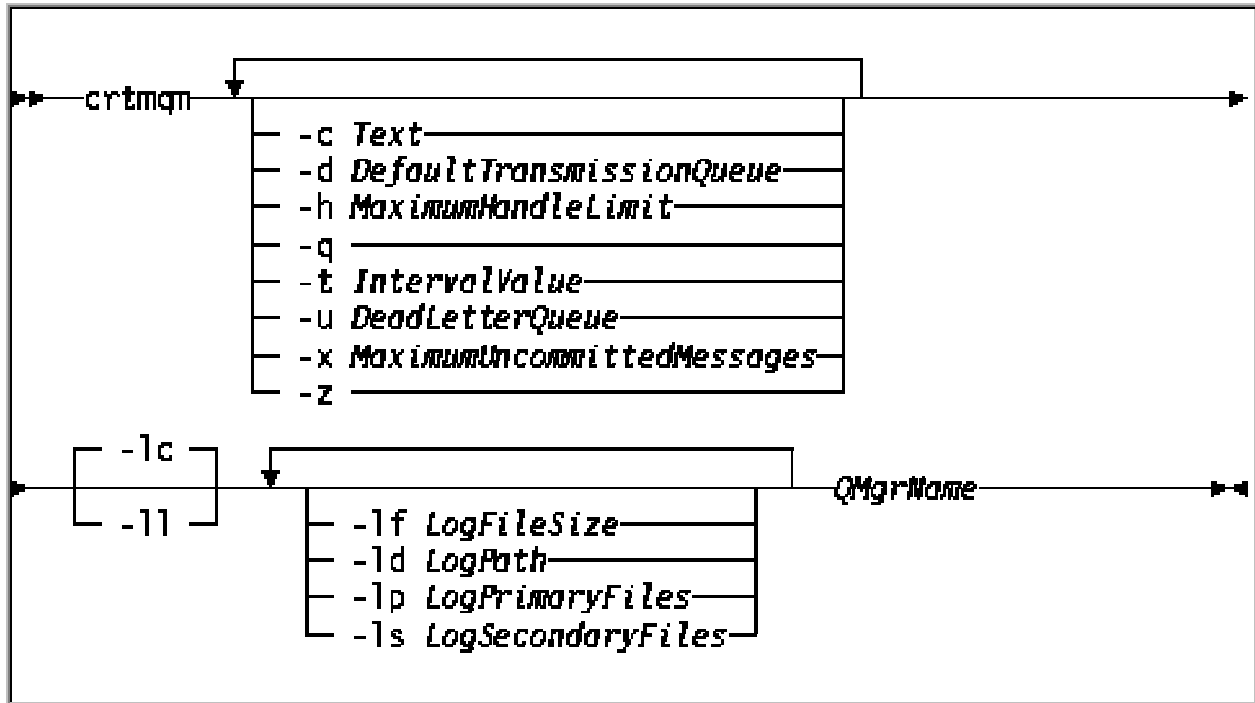
Flags/parameters

QMgrName

Specifies the name of the queue manager to be created. The name can contain up to 48 characters. This must be the last item in the command.

Optional parameters

-c Text

Specify some descriptive text for this queue manager. The default is all blanks.

You can use up to 64 characters. If special characters are required, the description must be enclosed in double quotes.

-d DefaultTransmissionQueue

Specifies the name of the local transmission queue that remote messages are placed on if a transmission queue is not explicitly defined for their destination. There is no default.

-h MaximumHandleLimit

Specifies the maximum number of handles that any one application can have open at the same time.

Specify a value in the range one through 999 999 999. The default value is 256.

-q

Specifies that this queue manager is to be made the default queue manager. The new queue manager replaces any existing queue manager as the default.

If you accidentally use this flag and wish to revert to an existing queue manager as the default queue manager, you can edit the `DefaultQueueManager` stanza in the MQSeries configuration file. See [Chapter 13. "Understanding configuration files"](#) for information about configuration files.

-t IntervalValue

Specifies the trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after the receipt of a trigger generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You may wish to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range zero through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be taken effectively means that triggering is disabled after the first trigger message. However, triggering can be reenabled by an application servicing the queue using an alter queue command to reset the trigger attribute.

-u DeadLetterQueue

Specifies the name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default if the attribute is omitted is no dead-letter queue.

-x MaximumUncommittedMessages

Specifies the maximum number of uncommitted messages under any one syncpoint. That is, the sum of:

- The number of messages that can be retrieved from queues.

- The number of messages that can be put on queues.

- Any trigger messages generated within this unit of work.

This limit does not apply to messages that are retrieved or put outside a syncpoint.

Specify a value in the range one through 10 000. The default value is 1000 uncommitted

messages.

`-z`

Error messages are to be suppressed.

This flag is normally used within MQSeries to suppress unwanted error messages. As use of this flag could result in loss of information, it is recommended that you do not use it when entering commands on a command line.

The following set of flags is used to define the logging to be used by the queue manager being created. For more information about logs, see <u>"Using the log for recovery"</u>.

`-lc`

Circular logging is to be used. This is the default logging method.

`-ll`

Linear logging is to be used.

`-lf LogFileSize`

Specifies the size of the log files in multiples of 4 KB. The minimum value is 64 KB, and the maximum is 16384 KB. The default value is 1024 KB, giving a default log size of 4 MB.

`-ld LogPath`

Specifies the directory to be used to hold the log files. The default is `/var/mqm/log`. The default can also be changed when MQSeries is customized.

User ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is done automatically if the logs files are in their default locations.

`-lp LogPrimaryFiles`

Specifies the number of primary log files to be allocated. The default value is 3, the minimum is 2, and the maximum is 62.

`-ls LogSecondaryFiles`

Specifies the number of secondary log files to be allocated. The default value is 2, the minimum is 1, and the maximum is 61.

Note: The total number of log files is restricted to 63, regardless of the number requested.

Return codes

0   Queue manager created

8   Queue manager already exists

49   Queue manager stopping

69   Storage not available

70   Queue space not available

71   Unexpected error

72   Queue manager name error

100   Log location invalid

111   Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect.

115   Invalid log size

Examples

1. This command creates a default queue manager named `Paint.queue.manager`, which is given a description of `Paint shop`. It also specifies that linear logging is to be used:

```
crtmqm -c "Paint shop" -ll -q Paint.queue.manager
```

2. This example requests a number of log files. 2 primary and 3 secondary log files are specified.

```
crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
```

3. In this example, another queue manager, `travel`, is created. The trigger interval is defined as 5000 milliseconds (or 5 seconds) and its dead-letter queue is specified as SYSTEM.DEAD.LETTER.QUEUE.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
```

Once a trigger event has been generated, further trigger events are disabled for five seconds.

Related commands

strmqm

    Start queue manager

endmqm

    End queue manager

dltmqm

    Delete queue manager

dltmqm (Delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager. All objects associated with this queue manager are also deleted. Before you can delete a queue manager you must end it using the **endmqm** command.

```
►►──dltmqm─────────────────QMgrName──────────────────────────►◄
              └─ -z ─┘
```

Flags/parameters

QMgrName

Specifies the name of the queue manager to be deleted.

Optional parameters

-z

Suppress error messages.

Return codes

0	Queue manager deleted
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid
112	Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect.

Examples

1. The following command deletes the queue manager `saturn.queue.manager`.

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

2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command.

```
dltmqm -z travel
```

Related commands

crtmqm

    Create queue manager

strmqm

    Start queue manager

endmqm

    End queue manager

dspmqaut (Display authority)

Purpose

Use the **dspmqaut** command to display the current authorizations to a specified object. Only one group may be specified.

If a user ID is a member of more than one group, examine the authorizations of each group to determine all the authorizations that apply to the user ID.

```
>>--dspmqaut------------------------------- -n ObjectName-- -t ObjectType--------->
                 |__-m QMgrName__|


>--+-------------------+--+-------------------------+--------------------------><
   |__-g GroupName_____|  |__-s ServiceComponent__|
   |__-p PrincipalName_|
```
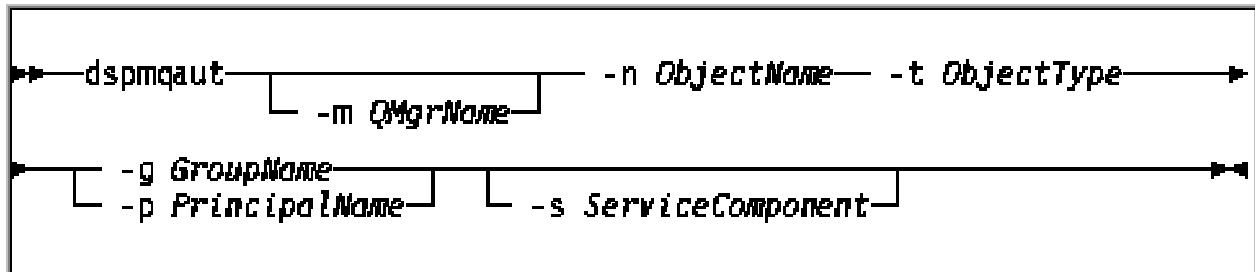
Flags/parameters

-t ObjectType

   Specifies the type of object on which the inquiry is to be made. Possible values are:

   queue or **q**   A queue or queues matching the object name parameter

   process or **prcs**   A process

   qmgr   A queue manager object

Optional parameters

-m QMgrName

   Specifies the name of the queue manager on which the inquiry is to be made.

-n ObjectName

   Specifies the name of the object on which the inquiry is to be made.
   This is a required parameter **UNLESS** it is the queue manager itself.
   You must specify the name of a queue manager, queue, or process definition.

-g GroupName

   Specifies the name of the user group on which the inquiry is to be made. You can only specify **one** name, which must be the name of an existing user group.

-p PrincipalName

   Specifies the name of a user whose authorizations to the specified object are to be displayed.

-s ServiceComponent

   This parameter only applies if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

Returned parameters

This command returns an authorization list, which can contain none, one, or more authorization parameters. Each authorization parameter returned means that any user ID in the specified group has the authority to perform the operation defined by that parameter.

Figure 31 shows the authorities that can be given to the different object types.

Figure 31. Security authorities from the dspmqaut command

Authority	Queue	Process	Qmgr
all	Yes	Yes	Yes
alladm	Yes	Yes	Yes
allmqi	Yes	Yes	Yes
altusr			Yes
browse	Yes		
chg	Yes	Yes	Yes
clr	Yes		
connect			Yes
crt	Yes	Yes	Yes
dlt	Yes	Yes	Yes
dsp	Yes	Yes	Yes
put	Yes		
inq	Yes	Yes	Yes
get	Yes		
passall	Yes		
passid	Yes		
set	Yes	Yes	Yes

setall	Yes		Yes
setid	Yes		Yes

The following list defines the authorizations associated with each parameter:

all

Use all operations relevant to the object.

alladm

Perform all administration operations relevant to the object.

allmqi

Use all MQI calls relevant to the object.

altusr

Specify an alternate user ID on an MQI call.

browse

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

chg

Change the attributes of the specified object, using the appropriate command set.

chgaut

Specify authorizations for other groups of users on the object, using the **setmqaut** command.

clr

Clear a queue (PCF command Clear queue only).

connect

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

crt

Create objects of the specified type, using the appropriate command set.

dlt

Delete the specified object, using the appropriate command set.

dsp

Display the attributes of the specified object, using the appropriate command set.

get

Retrieve a message from a queue by issuing an MQGET.

inq

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

passall

Pass all context.

passid

Pass the identity context.

put

Put a message on a specific queue by issuing an MQPUT.

set

Set attributes on a queue from the MQI by issuing an MQSET.

setall

Set all context on a queue.

setid

Set the identity context on a queue.

start

Start a queue manager, using the **strmqm** command.

stop

Stop a queue manager, using the **endmqm** command.

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands

- MQSC commands

- PCF commands

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing

147　　　　　Object type missing

148　　　　　Invalid object type

149　　　　　Entity name missing

Examples

The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

```
dspmqaut -m saturn.queue.manager -t qmgr -g staff
```

The results from this command are:

```
Entity staff has the following authorizations for object :
        get
        browse
        put
        inq
        set
        connect
        altusr
        passid
        passall
        setid
```

Related commands

Setmqaut

Set or reset authority

dspmqcsv (Display command server)

Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

The status can be one of the following:

- Starting

- Running

- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets

- Ending

- Stopped

```
►►──dspmqcsv──QMgrName────────────────────────────────────────────►◄
```

Flags/parameters

QMgrName

Specifies the name of the local queue manager for which the command server status is being requested.

Return codes

0   Command completed normally

10   Command completed with unexpected results

20   An error occurred during processing

Examples

The following command displays the status of the command server associated with venus.q.mgr:

```
dspmqcsv venus.q.mgr
```

Related commands

strmqcsv

Start a command server

endmqcsv

End a command server.

dspmqfls (Display MQSeries files)

Purpose

Use the **dspmqfls** command to display the real file system name for all MQSeries objects that match a specified criterion. You can use this command to identify the files associated with a particular MQSeries object. This is useful for backing up specific objects. See <u>"Understanding MQSeries file names"</u> for further information about name transformation.

```
>>──dspmqfls──┬──────────────┬──┬───────────┬──GenericObjName──────><
              └─-m QMgrName──┘  └─-t ObjType─┘
```
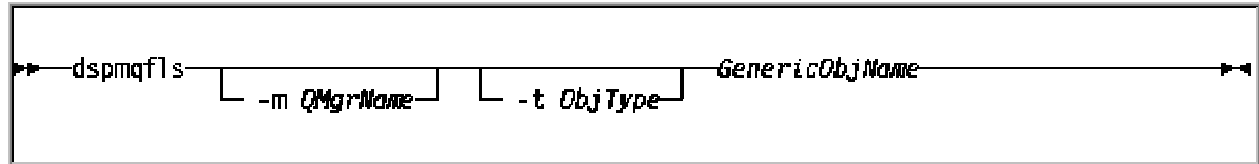
Flags/parameters

GenericObjName

> Specifies the name of the MQSeries object. The name is a string with no flag and is a required parameter. If the name is omitted an error is returned.

> This parameter supports a wild card character * at the end of the string.

Optional parameters

-m QMgrName

> Specifies the name of the queue manager for which files are to be examined. If omitted, the command operates on the default queue manager.

-t ObjType

> Specifies the MQSeries object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

* **or** all

> All object types. This is the default.

q **or** queue

> A queue or queues matching the object name parameter

ql **or** qlocal

> A local queue

qa **or** qalias

> An alias queue

qr **or** qremote

>A remote queue

qm **or** qmodel

>A model queue

qmgr

>A queue manager object

prcs **or** process

>A process

Note: You will need to prevent the shell from interpreting the meaning of special characters such as '*'. To accomplish this, use 'quoting'.

>ere are a number of ways of 'quoting' depending on your shell. For example, either single tation marks, double quotation marks, or a backslash, are used by some shells.

Return codes

0 Command completed normally

10 Command completed but not entirely as expected

20 An error occurred during processing

Examples

1. The following command displays the details of all objects with names beginning SYSTEM.ADMIN that are defined on the default queue manager.

```
dspmqfls SYSTEM.ADMIN*
```

2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

```
dspmqfls -m RADIUS -t prcs PROC*
```

dspmqtrc (Display MQSeries formatted trace output)

Purpose

Use the **dspmqtrc** command to display MQSeries formatted trace output.

```
>>--dspmqtrc------------------------------InputFileName------------------><
              |                        |
              |--t FormatTemplate------|
```
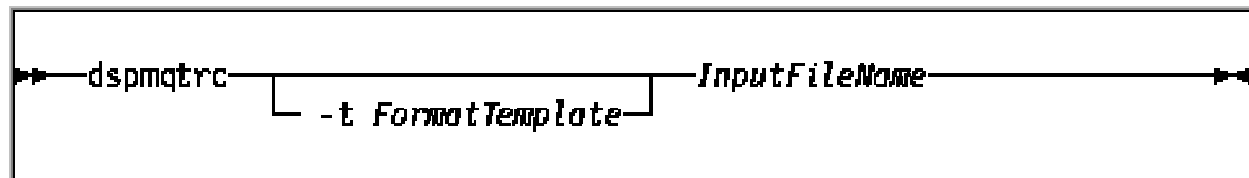
Flags/parameters

InputFileName

The name of the file containing the unformatted trace. For example
`/var/mqm/trace/AMQ12345.TRC..`

Optional parameters

-t FormatTemplate

is the name of the template file containing details of how to display the trace. The default value
is **mqmtop**`/lib/amqtrc.fmt`

Examples

This command ends tracing of data for a queue manager called `QM1`.

```
    endmqtrc -m QM1
```

Related commands

endmqtrc

End MQSeries trace

strmqtrc

Start MQSeries trace

dspmqtrn (Display MQSeries transactions)

Purpose

Use the **dspmqtrn** command to list the transactions that are in prepared status in a two-phase commit procedure and that are known to a queue manager (see warning below).

Each transaction is displayed as a transaction number (a human-readable transaction identifier), the transaction state, and the transaction ID. Transaction IDs can be up to 128 characters long, hence the need for a transaction number.
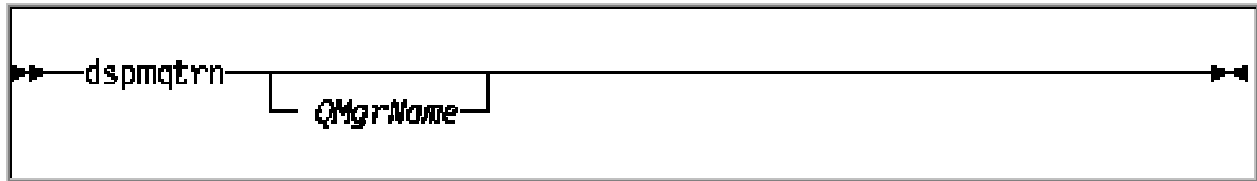
```
>>--dspmqtrn--------------------------------------------------><
            |                      |
            '-- QMgrName --------'
```

Warning. The only time that you can expect to use this command is if you are using an external transaction manager and are involved with two-phase commitment procedures. If you do not use two-phase commit, do not use this command. This command should be used only if the syncpoint manager has failed to resolve a transaction.

Optional parameters

QMgrName

Specifies the name of the queue manager whose transactions are to be examined. If omitted, the command operates on the default queue manager.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

rsvmqtrn

Resolve MQSeries transaction

endmqcsv (End command server)

Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.
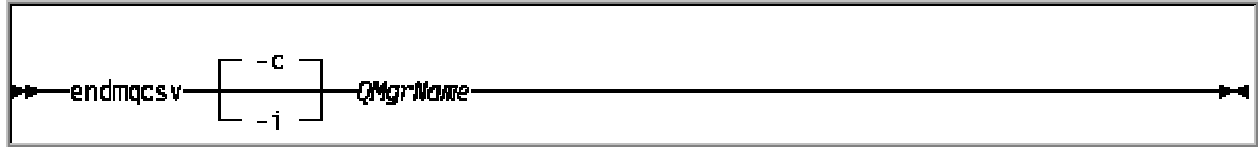
```
>>--endmqcsv--+--------+--QMgrName--------------------------------------><
              | -c |
              +--------+
              | -i |
```

Flags/parameters

QMgrName

Specifies the name of the queue manager for which the command server is to be ended.

Optional parameters

-c

Specifies that the command server is to be stopped in a controlled manner. The command server is allowed to complete the processing of any command message that it has already started. No new message is read from the command queue.

This is the default.

-i

Specifies that the command server is to be stopped immediately. Actions associated with a command message currently being processed may not be completed.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

1. The following command stops the command server on queue manager `saturn.queue.manager`:

```
endmqcsv  -c saturn.queue.manager
```

The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.

2. The following command stops the command server on queue manager `pluto` immediately:

```
endmqcsv -i pluto
```

Related commands

strmqcsv

Start a command server.

dspmqcsv

Display the status of a command server.

endmqm (End queue manager)

Purpose

Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Normal or quiesced shutdown

- Immediate shutdown

- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, you must stop it and then use the **dltmqm** (Delete queue manager) command.
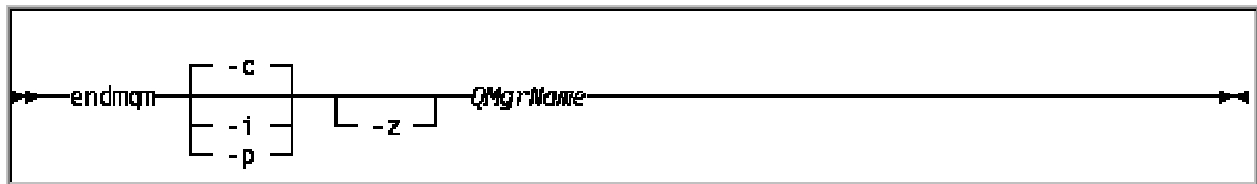
Flags/parameters

QMgrName

Specifies the name of the message queue manager to be stopped.

Optional parameters

-c

Controlled (or quiesced) shutdown. The queue manager stops but only after all applications have disconnected. Any MQI calls currently being processed are completed. This is the default.

-i

Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued are failed. Any incomplete units of work are rolled back when the queue manager is next started.

-p

Preemptive shutdown.

Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for MQI applications. All processes in the queue manager that fail to stop are terminated 30 seconds after the command is issued.

-z

Suppress error messages on the command.

Return codes

0	Queue manager ended.
3	Queue manager being created
16	Queue manager does not exist.
40	Queue manager not available.
49	Queue manager stopping.
69	Storage not available.
71	Unexpected error.
72	Queue manager name error.

Examples

The following examples show commands that end (stop) the specified queue managers.

1. This command ends the default queue manager in a controlled way. All applications currently connected are allowed to disconnect.

```
endmqm
```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

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

Related commands

crtmqm

Create a queue manager

strmqm

Start a queue manager

dltmqm

Delete a queue manager

endmqtrc (End MQSeries trace)

Purpose

Use the **endmqtrc** command to end tracing for the specified entity or all entities.
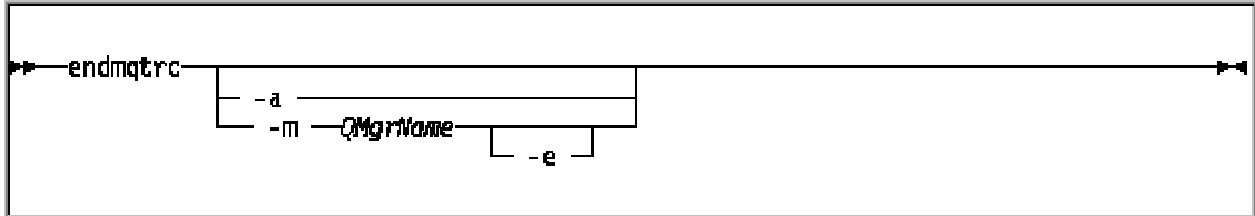
Optional parameters

-m *QMgrName*

Is the name of the queue manager for which tracing is to be ended, or is the literal value @SYSTEM used to end tracing of channels and the command server.

A maximum of one -m flag and associated queue manager name can be supplied on the command.

An -m flag and queue manager name can be specified on the same command as the -e flag.

-e

If this flag is specified, early tracing is ended.

-a

If this flag is specified all tracing is ended.

This flag *must* be specified alone.

Return codes

AMQ5611

This message is issued if arguments that are not valid are supplied to the command.

Examples

This command ends tracing of data for a queue manager called QM1.

```
endmqtrc -m QM1
```

Related commands

dspmqtrc

Display formatted trace output

strmqtrc

Start MQSeries trace

rcdmqimg (Record media image)

Purpose

Use the **rcdmqimg** command to write an image of an MQSeries object, or group of objects, to the log for use in media recovery. Use the associated command **rcrmqobj** to recreate the object from the image.

This command is used with an active queue manager. Further activity on the queue manager is logged so that, although the image becomes out of date, the log records reflect any changes to the object.

```
>>--rcdmqimg-----------------------------------------------t ObjectType--------->

              |__ -m --QMgrName__|  |__ -z __|

   >--GenericObjName--------------------------------------------------->
```
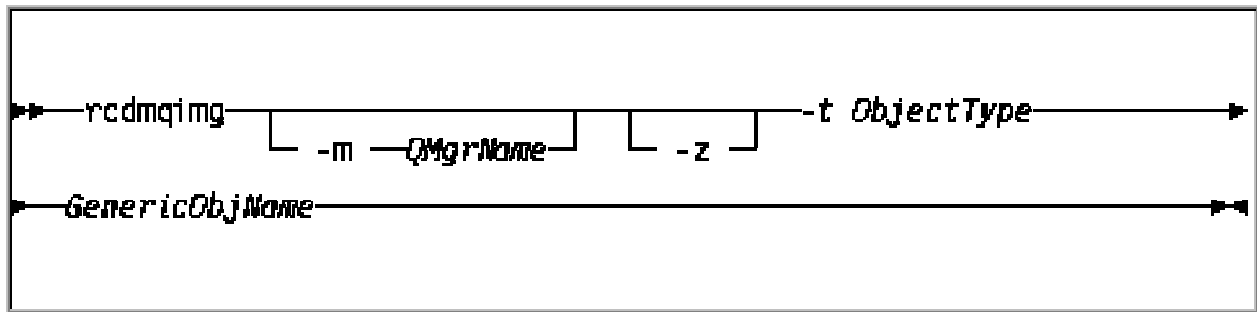
Flags/parameters

-t ObjectType

 Specifies the type of objects whose images are to be recorded. Valid object types are:

prcs or **process**

 Processes

q or **queue**

 All types of queue

ql or **qlocal**

 Local queues

qa or **qalias**

 Alias queues

qr or **qremote**

 Remote queues

qm or **qmodel**

 Model queues

qmgr

Queue manager object

* or all

All types of object

Note: You will need to prevent the shell from interpreting the meaning of special characters such as '*'. To accomplish this, use 'quoting'.

re are a number of ways of 'quoting' depending on your shell. For example, either single tation marks, double quotation marks, or a backslash, are used by some shells.

Optional parameters

-m QMgrName

Specifies the name of the queue manager for which images are to be recorded. If omitted, the command operates on the default queue manager.

-z

Suppress error messages.

GenericObjName

Specifies the name of the object that is to be recorded. This parameter may have a trailing asterisk to indicate that any objects with names matching the portion of the name prior to the asterisk are to be recorded. This parameter may be omitted if you are recording a queue manager.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
68	Media recovery is not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
131	Resource problem

132   Object damaged

135   Temporary object cannot be recorded

Examples

The following command records an image of the queue manager object `saturn.queue.manager` in the log.

```
mqimg -t qmgr -m saturn.queue.manager
```

Related commands

rcrmqobj

Recreates a queue manager object.

rcrmqobj (Recreate object)

Purpose

Use the **rcrmqobj** command to recreate an object, or group of objects, from their images contained in the log. Use the associated command, **rcdmqimg**, to record the object images to the log.

This command must be used on a running queue manager. All activity on the queue manager after the image was recorded is logged. To recreate an object you must replay the log to recreate events that occurred after the object image was captured.

```
►►──rcrmqobj──┬──────────────┬──┬──────┬──-t ObjectType───────►
              └─ -m ─QMgrName─┘  └─ -z ─┘
►── GenericObjName ──────────────────────────────────────►◄
```
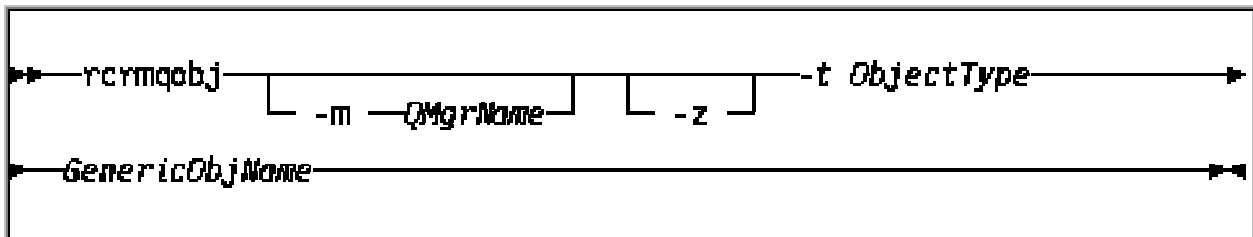
Flags/parameters

-t ObjectType

Specifies the type of objects to be recreated. Valid object types are:

prcs or **process**

Processes

q or **queue**

All types of queue

ql or **qlocal**

Local queues

qa or **qalias**

Alias queues

qr or **qremote**

Remote queues

qm or **qmodel**

Model queues

* or **all**

All the above

syncfile

> The channel synchronization file

Note: Using this flag causes the channel synchronization file to be regenerated for the queue manager specified. This is necessary because the file is not saved by the **rcdmqimg** command.

Note: You will need to prevent the shell from interpreting the meaning of special characters such as '*'. To accomplish this, use 'quoting'.

re a number of ways of 'quoting' depending on your shell. For example, either single
ɔn marks, double quotation marks, or a backslash, are used by some shells.

Optional parameters

-m *QMgrName*

> Specifies the name of the queue manager for which objects are to be recreated. If omitted, the command operates on the default queue manager.

-z

> Suppress error messages.

ObjectName

> Specifies the name of the object that is to be recreated. This parameter may have a trailing asterisk to indicate that any objects with names matching the portion of the name prior to the asterisk are to be recreated.

> If the object type is the channel synchronization file, the object name may be omitted. If an object name is supplied for this type, it is ignored.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
66	Media image not available
68	Media recovery is not supported
69	Storage not available
71	Unexpected error

72	Queue manager name error
119	User not authorized
128	No objects processed
135	Temporary object cannot be recovered
136	Object in use

Examples

1. The following command recreates all local queues for the default queue manager:

```
rcrmqobj -t ql *
```

2. The following command recreates all remote queues associated with queue manager store:

```
rcrmqobj -m store -t qr
```

Related commands

rcdmqimg

Records a queue manager object in the log.

rsvmqtrn (Resolve MQSeries transactions)

Purpose

Use the **rsvmqtrn** command to give a commit or backout decision to an in-doubt transaction.

Note: This command must be used only in situations where you are certain that the transaction will not be resolved by the normal protocols. Issuing this command may result in the loss of transactional integrity between resource managers for a distributed transaction.
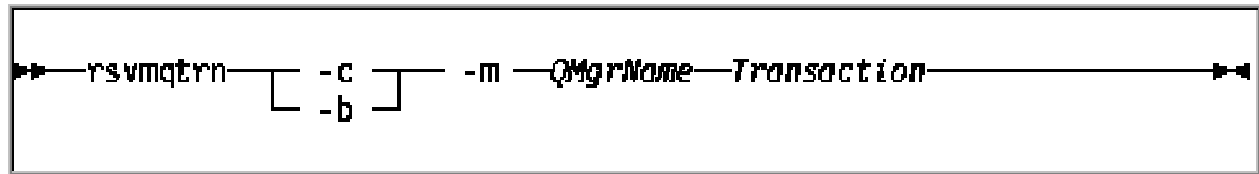
```
>>--rsvmqtrn--+--------+----m --QMgrName--Transaction--------------------><
              |  -c  |
              +--------+
                 -b
```

Warning. The only time that you can expect to use this command is if you are using an external transaction manager and are involved with two-phase commitment procedures.

If you do not use two-phase commit, do not use this command.

This command should be used only if the syncpoint manager has failed to resolve a transaction.

Flags/parameters

-c

Specifies a commit decision.

-b

Specifies a backout decision.

There is no default; you must supply one of these options.

-m QMgrName

Specifies the name of the queue manager whose transactions are to be resolved. The queue manager name must be specified.

Transaction

Specifies the transaction number of the transaction of interest. The number can be determined by using the **dspmqtrn** command to display all transactions on a queue manager that have been left in a prepared (in-doubt) state.

Return codes

0   Successful operation

36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

dspmqtrn

Display list of prepared transactions.

runmqchi (Run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to the *MQSeries Distributed Queuing Guide*.

```
>>--runmqchi-----------------------------------------------------------------><
                  |                          |   |                  |
                  +-- -q InitiationQName --+   +-- -m QMgrName --+
```
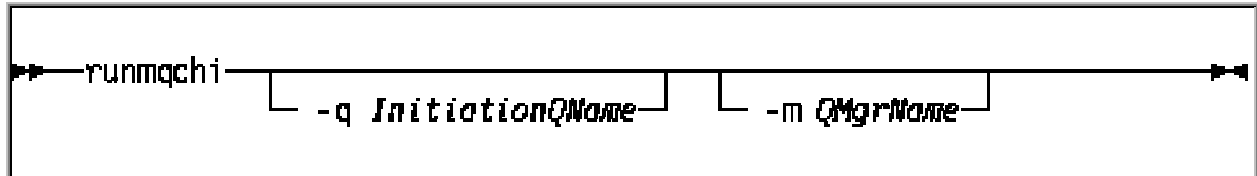
Optional parameters

-q InitiationQName

Specifies the name of the initiation queue to be processed by this channel initiator. If not specified, SYSTEM.CHANNEL.INITQ is used.

-m QMgrName

Specifies the name of the queue manager on which the initiation queue exists. If the name is omitted, the default queue manager is used.

Return codes

0   Command completed normally

10   Command completed with unexpected results

20   An error occurred during processing

If errors occur that result in return codes of either 10 or 20, you should review the queue manager error log that the channel is associated with for the error messages. You should also review the @SYSTEM error log as problems that occur before the channel is associated with the queue manager are recorded there. For more information about error logs, see "Error logs".

runmqchl (Run channel)

Purpose

Use the **runmqchl** command to run either a Sender (SDR) or a Requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

```
►►──runmqchl── -c ChannelName──────────────────────────────►◄
                      └─ -m QMgrName ─┘
```

Flags/parameters

-c ChannelName

Specifies the name of the channel to run.

Optional parameters

-m QMgrName

Specifies the name of the queue manager with which this channel is associated. If no name is specified, the default queue manager is used.

Return codes

0　　　　　Command completed normally

10　　　　Command completed with unexpected results

20　　　　An error occurred during processing

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages. You should also review the @SYSTEM error log because problems that occur before the channel is associated with the queue manager are recorded there.

runmqdlq (Run dead-letter queue handler)

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, a utility that you can run to monitor and handle messages on a dead-letter queue.

The dead-letter queue handler can be used to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from stdin. When the command is processed, the results and a summary are put into a report that is sent to stdout.

By taking stdin from the keyboard, you can enter **runmqdlq** rules interactively.

By redirecting the input from a file, you can apply a rules table to the specified queue. The rules table must contain at least one rule.

If the DLQ handler is used without redirecting stdin from a file (the rules table) the DLQ handler:

- Reads its input from the keyboard

- Does not start to process the named queue until it receives an end_of_file (ctrl-D) character.

For more information about rules tables and how to construct them, see ["The DLQ handler rules table"](#).

Syntax
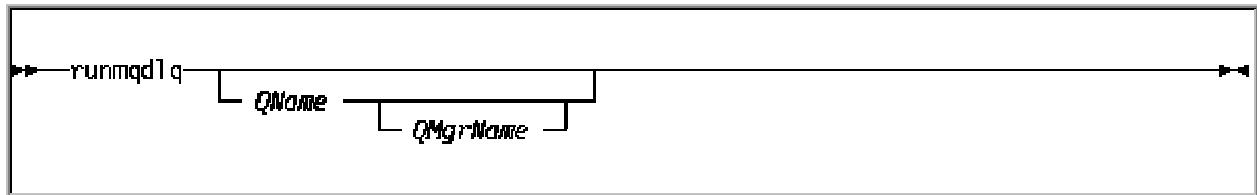
Optional parameters

The MQSC rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName

Specifies the name of the queue to be processed.

If no name is specified the dead letter queue defined for the local queue manager is used. If one or more blanks (' ') are used, the dead letter queue of the local queue manager is explicitly assigned.

A DLQ handler can be used to select particular messages on a dead-letter queue for special

processing. For example, this can be to redirect the messages to different dead-letter queues. Subsequent processing with another instance of the DLQ handler might then process the messages, according to a different rules table.

QMgrName

The name of the queue manager that owns the queue to be processed.

If no name is specified, the default queue manager for the installation is used. If one or more blanks (' ') are used, the default queue manager for this installation is explicitly assigned.

runmqsc (Run MQSeries commands)

Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration task, such as, defining, altering, or deleting a local queue object. MQSC commands and their syntax is described in the *MQSeries Command Reference.*

You can invoke the runmqsc command in three modes:

Verify mode

MQSC commands are verified but not actually run. An output report is generated indicating the success or failure of each command. This mode is only available on a local queue manager.

Direct mode

MQSC commands are sent directly to a local queue manager.

Indirect mode

MQSC commands are run on a remote queue manager. These commands are put on the command queue on a remote queue manager and are run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The runmqsc command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

Note: To run this command, you user ID must belong to user group `mqm`.
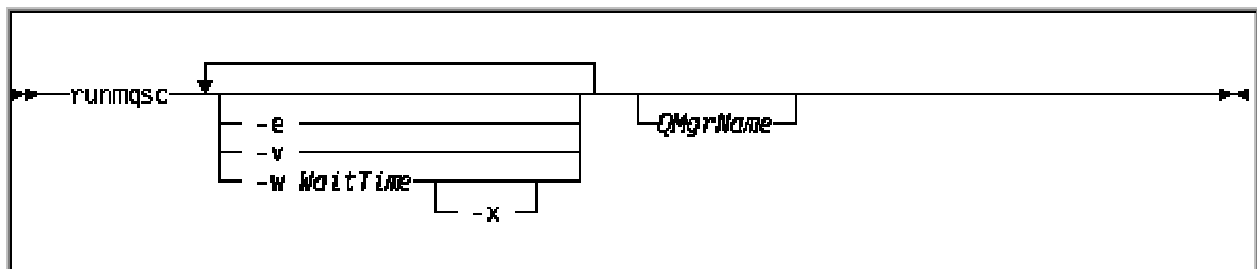
Optional parameters

-e

Prevents source text for the MQSC commands from being copied into a report. This is useful when you enter commands interactively.

-v

Specifies verification mode: verifies the specified commands without performing the actions. This mode is only available locally. The -w and -x flags are ignored if they specified at the same time.

-w WaitTime

Specifies indirect mode, that is, the MQSC commands are to be run on another queue manager. You must have the required channel and transmission queues set up for this. See "Preparing channels and transmission queues for remote administration" for more information.

WaitTime

Specifies the time, in seconds, the runmqsc waits for replies. Any replies received after this are discarded, however, the MQSC commands are still run. Specify a time between 1 and 999 999 seconds

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

Indirect mode operation is performed through the default queue manager.

This flag is ignored if the -v flag is specified.

-x

Specifies that the target queue manager is running under MVS/ESA. This flag applies only in indirect mode. The -w flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the MQSeries for MVS/ESA command queue.

QMgrName

Specifies the name of the target queue manager on which the MQSC are to be run. If omitted, the MQSC commands run on the default queue manager.

Return codes

00   MQSC command file processed successfully.

10   MQSC command file processed with errors--report contains reasons for failing commands.

20   Error--MQSC command file not run.

Examples

1. Type in this command at the IRIX command prompt:

```
runmqsc
```

```

```

Now you can type MQSC commands directly at the IRIX command prompt. No queue manager name was specified, therefore, the MQSC commands are processed on the default queue manager.

2. Use this command to specify that MQSC commands are verified only:

```
runmqsc -v BANK < /u/users/commfile.in
```

Verifies the MQSC command file `commfile.in` in directory `/u/users`. The queue manager name is `BANK`. The output is displayed in the current window.

3. This command runs the MQSC command file `/var/mqm/mqsc/mqscfile.in.` against the default queue manager.

```
runmqsc < /var/mqm/mqsc/mqscfile.in > /var/mqm/mqsc/mqscfile.out
```

In this example, the output is directed to file `/var/mqm/mqsc/mqscfile.out`.

runmqtmc (Start client trigger monitor)

Purpose

Use the **runmqtmc** command to invoke a trigger monitor for a client. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide*.

Note: This command is available **ONLY** on OS/2 and AIX clients.

```
>>--runmqtmc---------------------------------------------------------><
                 └─ -m QMgrName ─┘    └─ -q InitiationQName ─┘
```
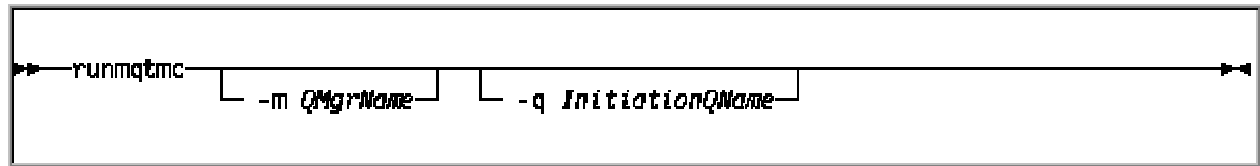
Optional parameters

-m QMgrName

Specifies the name of the queue manager on which the client trigger monitor operates. If omitted, the client trigger monitor operates on the default queue manager.

-q InitiationQName

Specifies the name of the initiation queue to be processed. If omitted, **SYSTEM.DEFAULT.INITIATION.QUEUE** is used.

Return codes

0   Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.

10   Client trigger monitor interrupted by an error.

20   Error--client trigger monitor not run.

runmqtrm (Start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide.*

```
►►──runmqtrm──┬──────────────┬──┬──────────────────────┬──►◄
              └─ -m QMgrName ─┘  └─ -q InitiationQName ─┘
```
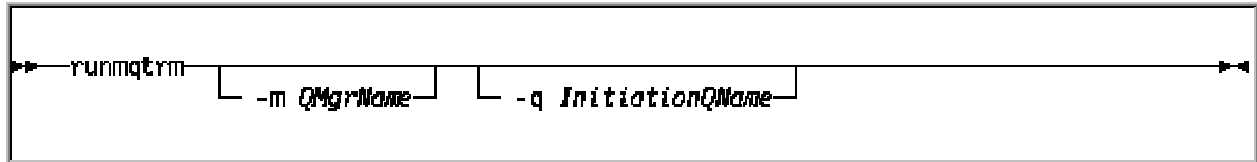
Optional parameters

-m QMgrName

Specifies the name of the queue manager on which the trigger monitor operates. If omitted, the trigger monitor operates on the default queue manager.

-q InitiationQName

Specifies the name of the initiation queue to be processed. If omitted, **SYSTEM.DEFAULT.INITIATION.QUEUE** is used.

Return codes

0   Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.

10   Trigger monitor interrupted by an error.

20   Error--trigger monitor not run.

setmqaut (Set/reset authority)

Purpose

Use the **setmqaut** command to change the authorizations to an object or to a class of objects.
Authorizations can be granted or revoked to any number of principals or groups.

Syntax

```
►►──setmqaut── -m QMgrName── -n ObjectName── -t ObjectType───────────►

   ┌──────────────────────────┐  ┌──────────────────────┐
►──┤                          ├──┤  ┌─ -p ─ PrincipalName ┤──────────►
      └─ -s ServiceComponent ─┘     └─ -g ─ GroupName ─────┘

   ┌────────────────────────────────────────┐
►──┤  ┌─ MQI authorizations ─────────────┤ ├──────────────────────►◄
      ├─ Context authorizations ──────┤ ─┤
      ├─ Administration authorizations ┤ ─┤
      └─ Generic authorizations ──────┤ ─┘
```

MQI authorizations:

```
   ┌──────────────────────┐
├──┤  ┌─ +altusr ─────────┤ ├───────────────────────────────────────
      ├─ -altusr ─┤
      ├─ +browse ─┤
      ├─ -browse ─┤
      ├─ +connect ┤
      ├─ -connect ┤
      ├─ +get ────┤
      ├─ -get ────┤
      ├─ +inq ────┤
      ├─ -inq ────┤
      ├─ +put ────┤
      ├─ -put ────┤
      ├─ +set ────┤
      └─ -set ────┘
```

Context authorizations:

```
   ┌──────────────────────┐
├──┤  ┌─ +passall ────────┤ ├───────────────────────────────────────
      ├─ -passall ┤
      ├─ +passid ─┤
      ├─ -passid ─┤
      ├─ +setall ─┤
      ├─ -setall ─┤
      ├─ +setid ──┤
      └─ -setid ──┘
```

Administration authorizations:

```
   ┌──────────────────────┐
├──┤  ┌─ +chg ────────────┤ ├───────────────────────────────────────
      ├─ -chg ─┤
      ├─ +clr ─┤
      ├─ -clr ─┤
      ├─ +cpy ─┤
      ├─ -cpy ─┤
      ├─ +crt ─┤
      ├─ -crt ─┤
      ├─ +dlt ─┤
      ├─ -dlt ─┤
      ├─ +dsp ─┤
      └─ -dsp ─┘
```

Generic authorizations:

```
   ┌──────────────────────┐
├──┤  ┌─ +all ────────────┤ ├───────────────────────────────────────
      ├─ -all ───┤
      ├─ +alladm ┤
      ├─ -alladm ┤
      ├─ +allmqi ┤
      └─ -allmqi ┘
```

Description

You can use this command both to *set* an authorization, that is, give a user group or principal permission to perform an operation, and to *reset* an authorization, that is, remove the permission to perform an operation. You must specify the user groups and principals to which the authorizations apply and also the queue manager, object type, and object name of the object. You can specify any number of groups and principals in a single command.

Caution:

If you specify a set of authorizations for a principal, the same authorizations are given to all principals in the same primary group.

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls

- Authorizations for MQI context

- Authorizations for issuing commands for administration tasks

- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by '+' or '-'. For example, if you include **+put** in the authorization list, you are giving authority to issue MQPUTs against a queue. Alternatively, if you include -**put** in the authorization list, you are removing the authorization to issue MQPUTs.

Authorizations can be specified in any order provided that they do not clash. For example, specifying **allmqi** with **set** causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

Flags/parameters

-m QMgrName

Specifies the name of the queue manager of the object for which the authorizations are to be changed. The name can contain up to 48 characters.

-t ObjectType

Specifies the type of object for which the authorizations are to be changed.

Possible values are:

- **q** or **queue**

- **prcs** or **process**

- **qmgr**

Optional parameters

-n *ObjectName*

Specifies the name of the object for which the authorizations are to be changed.

This is a required parameter **UNLESS** it is the queue manager itself. You must specify the name of a queue manager, queue, or process, but must not use a generic name.

-p *PrincipalName*

Specifies the name of the principal for which the authorizations are to be changed.

You must have at least one principal or one group.

-g *GroupName*

Specifies the name of the user group whose authorizations are to be changed. You can specify more than one group name, but each name must be prefixed by the '-g' flag.

-s *ServiceComponent*

This parameter applies only if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

Authorizations

Specifies the authorizations to be given or removed. Each item in the list is prefixed by a '+' indicating that authority is to be given, or a '-', indicating that authorization is to be removed. For example, to give authority to issue an MQPUT call from the MQI, specify **+put** in the list. To remove authority to issue an **mqput** call, specify **-put.**

Figure 32 shows the authorities that can be given to the different object types.

Figure 32. Specifying authorizations for different object types

Authority	Queue	Process	Qmgr
all	Yes	Yes	Yes
alladm	Yes	Yes	Yes
allmqi	Yes	Yes	Yes
altusr			Yes
browse	Yes		
chg	Yes	Yes	Yes
clr	Yes		
connect			Yes
crt	Yes	Yes	Yes
dlt	Yes	Yes	Yes
dsp	Yes	Yes	Yes
put	Yes		
inq	Yes	Yes	Yes
get	Yes		
passall	Yes		
passid	Yes		
set	Yes	Yes	Yes
setall	Yes		Yes
setid	Yes		Yes

Authorizations for MQI calls

get

Retrieve a message from a queue by issuing an MQGET.

browse

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

put

Put a message on a specific queue by issuing an MQPUT.

inq

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

set

Set attributes on a queue from the MQI by issuing an MQSET.

connect

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

altusr

Use an alternate user ID in a message.

See the *MQSeries Application Programming Guide* for more information about alternate user IDs.

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

Authorizations for context

passid

Pass identity context on the specified queue. The identity context is the same as that of the request.

passall

Pass all context on the specified queue. All the context fields are copied from the original request.

setid

Set identity context on the specified queue. This is used by special system utilities.

setall

Set all context on the specified queue. This is used by special system utilities.

Authorizations for commands

crt

Create objects of the specified type.

dlt

Delete the specified object.

chg

Change the attributes of the specified object,

dsp

Display the attributes of the specified object.

cpy

Copy the attributes of the specified object (PCF Copy commands only).

clr

Clear the specified queue (PCF Clear queue command only).

Authorizations for generic operations

allmqi

Use all MQI calls applicable to the object.

alladm

Perform all administration operations applicable to the object.

all

Use all operations applicable to the object.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error

133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing
150	Authorization specification missing
151	Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue *orange.queue* on queue manager *saturn.queue.manager*.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g tango +inq +alladm
```

The authorizations are being given to user group tango and the associated authorization list specifies that user group tango:

- Can issue **mqinq** calls.

- Has authority to perform all administration operations on that object.

2. In this example, the authorization list specifies that user group foxy:

- Cannot issue any calls from the MQI to the specified queue.

- Has authority to perform all administration operations on the specified queue.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g foxy -allmqi +alladm
```

Related commands

dspmqaut

Display authority

strmqcsv (Start command server)

Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables MQSeries to process commands sent to the command queue.

```
►►──strmqcsv──QMgrName──────────────────────────────────────►◄
```

Flags/parameters

QMgrName

Specifies the name of the queue manager for which the command server is to be started.

Return codes

0	Command completed normally
10	Command completed with unexpected results.
20	An error occurred during processing

Examples

The following command starts a command server for queue manager `earth`:

```
strmqcsv earth
```

Related commands

endmqcsv

End a command server.

dspmqcsv

Display the status of a command server.

strmqm (Start queue manager)

Purpose

Use the **strmqm** command to start a local queue manager.

```
>>──strmqm─────────────────────────────────────────><
            └─ -z ─┘      └─QMgrName─┘
```
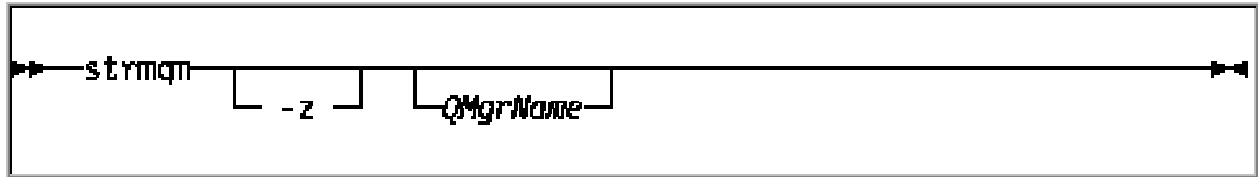
Optional parameters

QMgrName

Specifies the name of a local queue manager to be started. If omitted, the default queue manager is started.

-z

Suppress error messages.

This flag is used within MQSeries to suppress unwanted error messages. Because using this flag could result in loss of information, you should not use it when entering commands on a command line.

Return codes

0	Queue manager started
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
23	Log not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid

Examples

The following command starts the queue manager account:

```
    strmqm account
```

Related commands

crtmqm

    Create a queue manager

dltmqm

    Delete a queue manager

endmqm

    End a queue manager .

strmqtrc (Start MQSeries trace)

Purpose

Use the **strmqtrc** command to enable tracing. This command can be run whether tracing is enabled or not. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.
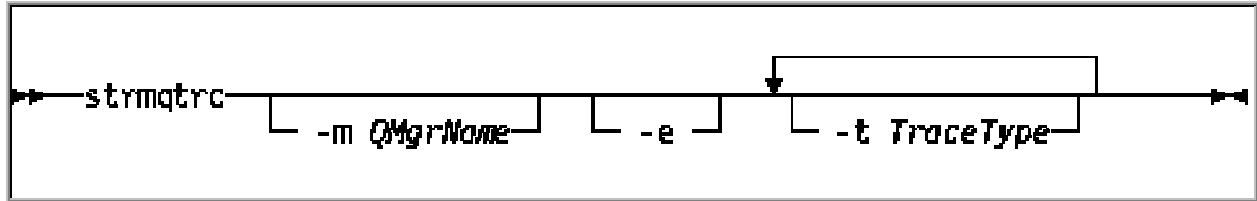
Optional parameters

-m QMgrName

> Is the name of the queue manager to be traced, or is the literal value @SYSTEM used to trace channels and the command server. If the -m flag is omitted, no tracing of queue managers or @SYSTEM occurs. In particular, you cannot trace the default queue manager by omitting the -m flag and queue manager name.

> An -m flag and queue manager name can be specified on the same command as the -e flag. If more than one trace specification applies to a given entity being traced, the actual trace includes all of the specified options.

> The queue manager, specified with the -m flag, does not have to be running or even to exist. Consequently, it is possible to trace the creation or startup of a queue manager.

> It is an error to omit the -m flag and queue manager name, unless the -e flag is specified.

-e

> If this flag is specified, early tracing is requested. This involves trace information being written, before the processes know to which MQSeries component they belong. Any process, belonging to any component of any queue manager, traces its early processing if this flag is specified. The default, if this flag is not specified, is not to perform early tracing.

-t TraceType

> Defines which points during processing can be traced. If this flag is omitted, all trace points are enabled and a full trace generated.

> Alternatively, one or more of the options in the following list can be supplied.

Note: If multiple trace types are supplied, each *must* have its own -t flag. Any number of -t flags can be specified, as long as each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

all

Output data for every trace point in the system. This is also the default if the -t flag is not specified.

api

Output data for trace points associated with the MQI and major queue manager components.

comms

Output data for trace points associated with data flowing over communications networks

csflows

Output data for trace points associated with processing flow in common services.

lqmflows

Output data for trace points associated with processing flow in the local queue manager.

remoteflows

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

otherflows

Output data for trace points associated with processing flow in other components.

csdata

Output data for trace points associated with internal data buffers in common services

lqmdata

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

remotedata

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

otherdata

Output data for trace points associated with internal data buffers in other components.

versiondata

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

commentary

Output data for trace points associated with comments in the MQSeries components.

Return codes

AMQ7024

This message is issued if arguments that are not valid are supplied to the command.

AMQ8304

The maximum number of nine concurrent traces is already running.

Examples

This command enables tracing of data from common services and the local queue manager, for a queue manager called QM1.

```
strmqtrc -m QM1 -t csdata -t lqmdata
```

Related commands

dspmqtrc

Display formatted trace output

endmqtrc

End MQSeries trace.

Part 3. Appendixes

Appendix A. MQSeries for IRIX at a glance

Part numbers

- MQSeries for IRIX Version 2 Release 1, part number 98MS04221-01-001.

Machine requirements

- MQSeries Servers:
  Any SGI IRIX supported system

  Minimum system disk space 20MB

- MQSeries Clients:
  Client code for IRIX, OS/2, DOS, and Windows 3.1 workstations is distributed with the
  server code.

Client software provides a remote interface to a LAN server. Client software can reside at the server
- or at a file server - and be dynamically copied to the client for use, or it can reside on the client
disk-space.

Full details of all client operating environments are given in the *MQSeries Clients* book.

Software requirements

Minimum supported levels are shown. Later levels, if any, will be supported unless otherwise stated.

- SGI IRIX version 6.2, 6.3, 6.4 or 6.5.

Connectivity

Network protocols supported are TCP/IP.

Any communications hardware supporting TCP/IP

Compilers supported for MQSeries applications

- MIPSpro C or C++ compilers version 7.2.1 or later.

- Micro Focus Object COBOL version 4.x or later.

Options

The following transaction processing monitor (coordination through X/Open XA interface) is

supported by MQSeries for IRIX:

- BEA TUXEDO

Delivery

MQSeries for IRIX V2 is supplied on CD-ROM.

Installation

MQSeries for IRIX is installed with the "/usr/sbin/swmgr" program, and takes approximately 5 minutes to install.

Appendix B. System defaults

The sample MQSC command file `amqscoma.tst` contains definitions for the MQSeries for IRIX default and system objects. The default object definitions contain a complete set of attributes for that object. When you create an object, its attributes are inherited from the default object, except the ones you explicitly specify. The system objects are required for the operation of a queue manager or channel. Figure 33 lists the objects defined in `amqscoma.tst`.

You should create these objects for each queue manager on a given node. To create these objects, see "Running the supplied MQSC command files".

Figure 33. Objects included in amqscoma.tst

Object name	Description
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.DEAD.LETTER.QUEUE	Sample dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.PROCESS	Default process definition.
SYSTEM.DEF.SENDER	Default sender channel.
SYSTEM.DEF.SERVER	Default server channel.
SYSTEM.DEF.RECEIVER	Default receiver channel.
SYSTEM.DEF.REQUESTER	Default requester channel.
SYSTEM.DEF.SVRCONN	Default server connection channel.
SYSTEM.DEF.CLNTCONN	Default client connection channel.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	Channel synchronization queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.

SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands, and PCF commands.
SYSTEM.MQSC.REPLY.QUEUE	MQSC reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channel events.

Appendix C. Directory structure

shows the general layout of the data and log directories associated with a specific queue manager. The directories shown apply to the default installation. If you change this, the locations of the files and directories will be modified accordingly. For information about the location of the product files, see .

Figure 34. Default directory structure after a queue manager has been started

```
/var/mqm/
       — mqs.ini
       — qmgrs/ ——— @ SYSTEM/ ——— errors/
                 └── qmname/ ——— amqalchk.fil
                                — auth/ ———————————————————— @ aclass
                                — dce/                      — procdef/ ——— @ class
                                — errors/ ——— AMQERR01.LOG  — qmanager/ —┬ @ class
                                — plugcomp/   AMQERR02.LOG                └ self
                                — procdef/    AMQERR03.LOG  — queues/ ——— @ class
                                — qmanager/ —┬ QMANAGER     — QAADMIN
                                — qm.ini     └ QMQMOBJCAT
                                — queues/
                                — startprm/
                                — esem/
                                — isem/
                                — msem/
                                — shmem/ ——— PerQUEUE/
                                — ssem/
                                — @ ipcc/ ——————————————————— AMQCLCHL.TAB
                                                             — AMQRFCDA.DAT
                                                             — AMQRSYNA.DAT
       — log/ ——— qmname/ —┬ amqhlctl.lfh    — esem/
                           └ active/ —┬ S0000000.LOG — isem/
                                      — S0000001.LOG — msem/
                                      └ S0000002.LOG — shmem/ ——— PerQUEUE/
                                                     — ssem/
```

In [Figure 34](#), the layout is representative of MQSeries after a queue manager has been in use for some time. The actual structure that you have depends on which operations have occurred on the queue manager.

By default, the following directories and files located in the directory `/var/mqm/qmgrs/qmname/`.

amqalchk.fil

Checkpoint file containing information about last checkpoint.

auth/

This directory contains subdirectories and files associated with authority.

@aclass

This file contains the authority stanzas for all classes.

procdef/

This directory contains a file for each process definition. Each file contains the authority stanzas for the associated process definition.

@class

This file contains the authority stanzas for the process definition class.

qmanager/

@class

This file contains the authority stanzas for the queue manager class.

self

This file contains the authority stanzas for the queue manager object.

queues/

This directory contains a file for each queue. Each file contains the authority stanzas for the associated queue.

@class

This file contains the authority stanzas for the queue class.

QAADMIN

File used internally for controlling authorizations.

dce/

Empty directory reserved for use by DCE support.

errors/

The operator message files; from newest to oldest.
AMQERR01.LOG
AMQERR02.LOG
AMQERR03.LOG

plugcomp/

Empty directory reserved for use by installable services.

procdef/

Each MQSeries process definition is associated with a file in this directory. The file name matches the process definition name--subject to certain restrictions, see ["Understanding MQSeries file names"](#).

qmanager/

QMANAGER

> The queue manager object.

QMQMOBJCAT

> The object catalogue containing the list of all MQSeries objects--used internally.

qm.ini

> Queue manager configuration file.

queues/

> Each queue has a directory in here containing a single file called 'q'.
> The file name matches the queue name--subject to certain restrictions, see <u>"Understanding MQSeries file names"</u>.

startprm/

> Directory containing temporary files used internally.

esem/

> Directories containing files used internally.

isem/

msem/

shmem/

PerQUEUE/

> Directory containing files used internally.

ssem/

> Directory containing files used internally.

@ipcc/

AMQCLCHL.TAB

> Client channel table file.

AMQRFCDA.DAT

> Channel table file.

AMQRSYNA.DAT

> Channel synchronization file.

esem/

> Directories containing files used internally.

msem/

ssem/

isem/

shmem/

PerQUEUE/

  Directory containing files used internally.

Queue manager log directory structure

By default, the following directories and files are found in `/var/mqm/log/qmname/`.

The following subdirectories and files exist after you have installed MQSeries, created and started a queue manager, and have been using that queue manager for some time.

amqhlctl.lfh

  Log control file.

active/

  This directory contains the log files, numbered as follows:
  S0000000.LOG
  S0000001.LOG
  S0000002.LOG
  ... and so on.

Appendix D. Sample MQI programs and MQSC files

MQSeries for IRIX provides a set of short sample MQI programs, and MQSC command files. You can use these directly or modify them for experimental purposes.

MQSC command file samples

Figure 35 lists the MQSC command file samples. These are simply ASCII text files containing MQSC commands. You can invoke the **runmqsc** command against each file in turn to create the objects specified in the file. See "Running the supplied MQSC command files".

By default, these files are located in directory **mqmtop**/samp.

Figure 35. MQSC command files

File name	Purpose
Amqscoma.tst	Contains definitions of the default and system objects. These are required. Any object you define inherits attributes from the default objects except the attributes that you specify. The system objects are support the operation of a queue manager.
amqscos0.tst	Creates a set of MQI objects for use with the C program samples.

Figure 36 lists the sample MQI source files. By default, the source files are located in directory **mqmtop**/samp and the compiled versions in directory **mqmtop**/bin. To find out more about what the programs do and how to use them, see the *MQSeries Application Programming Guide* manual.

Figure 36. Sample programs - source files

C	Purpose
amqsbcg0.c	Reads and then outputs both the message descriptor and message context fields of all the messages on a specified queue.
amqsecha.c	Echoes a message from a message queue to the reply-to queue. Can be run as a triggered application program.
amqsgbr0.c	Writes messages from a queue to Stdout, leaving the messages on the queue. Uses **MQGET** with the browse option.
amqsget0.c	Removes the messages from the named queue (using **MQGET**) and writes them

	to StdOut.
amqsinqa.c	Reads the triggered queue; each request read as a queue name; responds with information about that queue.
amqsput0.c	Copies StdIn to a message and then puts this message on a specified queue.
amqsreq0.c	Puts request messages on a specified queue and then displays the reply messages.
amqsseta.c	Inhibits puts on a named queue and responds with a statement of the result. Runs as a triggered application.
amqstrg0.c	A trigger monitor that reads a named initiation queue and then starts the program associated with each trigger message. Provides a subset of the full triggering function of the supplied **runmqtrm** command.
amqsvfcx.c	A sample C skeleton of a Data Conversion exit routine.
Note: You can create the objects required by these samples using the MQSC command file `amqscos0.tst`.	

Miscellaneous tools

These tool files are provided to support the formatter and code conversion.

Figure 37. Miscellaneous files

File name	Location	Purpose
amqtrc.fmt	**mqmtop**/lib	Defines MQSeries trace formats.

Appendix E. Support for different codesets on MQSeries for IRIX

MQSeries for IRIX supports most of the codesets used by the locales - that is, the subsets of the user's environment which define the conventions for a specific culture - that are provided as standard on MQSeries for IRIX.

Note that to install most of these locales on an MQSeries for IRIX system the Language Supplement (LS) for the operating system must be installed. If the locale is not set, the value of the LANG environment variable is used. If neither the locale nor LANG environment variable is set the CCSID used is 819 - the ISO 8859-1 codeset.

The CCSID (Coded Character Set Identifier) used in MQSeries to identify the codeset used for the message and message header data is obtained by analyzing the LC_CTYPE environment variable.

Figure 38 shows the locales and the CCSIDs that are registered for the codeset used by the locale.

Figure 38. Locales and CCSIDs

Locale	Language	codeset	CCSID
C	English	ISO 8859-1	819
Da	Danish	ISO 8859-1	819
Da_DK	Danish	ISO 8859-1	819
Da_DK.850	Danish	PC 850	850
Da_DK.865	Danish	PC 865	865
De	German	ISO 8859-1	819
De_DE	German	ISO 8859-1	819
De_DE.437	German	PC 437	437
De_DE.850	German	PC 850	850
De_AT	German - Austria	ISO 8859-1	819
De_AT.437	German - Austria	PC 437	437
De_AT.850	German - Austria	PC 850	850

De_CH	German - Switzerland	ISO 8859-1	819
De_CH.437	German - Switzerland	PC 437	437
De_CH.850	German - Switzerland	PC 850	850
el	Greek	ISO 8859-7	813
el_GR	Greek	ISO 8859-7	813
en	English - United Kingdom	ISO 8859-1	819
en_GB	English - United Kingdom	ISO 8859-1	819
en_GB.437	English - United Kingdom	PC 437	437
en_GB.850	English - United Kingdom	PC 850	850
en_AU	English - Australia	ISO 8859-1	819
en_AU.437	English - Australia	PC 437	437
en_AU.850	English - Australia	PC 850	850
en_CA	English - Canada	ISO 8859-1	819
en_CA.850	English - Canada	PC 850	850
en_CA.863	English - Canada	PC 850	850
en_US	English - USA	ISO 8859-1	819
en_US.437	English - USA	PC 437	437
en_US.850	English - USA	PC 850	850
es	Spanish	ISO 8859-1	819
es_ES	Spanish	ISO 8859-1	819
es_ES.437	Spanish	PC 437	437
es_ES.850	Spanish	PC 850	850
fi	Finnish	ISO 8859-1	819
fi_FI	Finnish	ISO 8859-1	819
fi_FI.437	Finnish	PC 437	437

fi_FI.850	Finnish	PC 850	850
fr	French - France	ISO 8859-1	819
fr_FR	French - France	ISO 8859-1	819
fr_FR.437	French - France	PC 437	437
fr_FR.850	French - France	PC 850	850
fr_BE	French - Belgium	ISO 8859-1	819
fr_BE.437	French - Belgium	PC 437	437
fr_BE.850	French - Belgium	PC 850	850
fr_CA	French - Canada	ISO 8859-1	819
fr_CA.850	French - Canada	PC 850	850
fr_CA.863	French - Canada	PC 863	863
fr_CH	French - Switzerland	ISO 8859-1	819
fr_CH.437	French - Switzerland	PC 437	437
fr_CH.850	French - Switzerland	PC 850	850
is	Icelandic	ISO 8859-1	819
is_IS	Icelandic	ISO 8859-1	819
is_IS.850	Icelandic	PC 850	850
it	Italian - Italy	ISO 8859-1	819
it_IT	Italian - Italy	ISO 8859-1	819
it_IT.437	Italian - Italy	PC 437	437
it_IT.850	Italian - Italy	PC 850	850
it_CH	Italian - Switzerland	ISO 8859-1	819
it_CH.437	Italian - Switzerland	PC 437	437
it_CH.850	Italian - Switzerland	PC 850	850
nl	Dutch - Netherlands	ISO 8859-1	819

nl_NL	Dutch - Netherlands	ISO 8859-1	819
nl_NL.437	Dutch - Netherlands	PC 437	437
nl_NL.850	Dutch - Netherlands	PC 850	850
nl_BE	Dutch - Belgium	ISO 8859-1	819
nl_BE.437	Dutch - Belgium	PC 437	437
nl_BE.850	Dutch - Belgium	PC 850	850
no	Norwegian	ISO 8859-1	819
no_NO	Norwegian	ISO 8859-1	819
no_NO.850	Norwegian	PC 850	850
no_NO.865	Norwegian	PC 865	865
pl	Polish	ISO 8859-2	912
pl_PL	Polish	ISO 8859-2	912
POSIX	English	ISO 8859-1	819
pt	Portuguese	ISO 8859-1	819
pt_PT	Portuguese	ISO 8859-1	819
pt_PT.850	Portuguese	PC 850	850
pt_PT.860	Portuguese	PC 860	860
ru	Russian	ISO 8859-5	915
ru_SU	Russian	ISO 8859-5	915
sh	Serbocroatian	ISO 8859-2	912
sh_YU	Serbocroatian	ISO 8859-2	912
sv	Swedish	ISO 8859-1	819
sv_SE	Swedish	ISO 8859-1	819
sv_SE.437	Swedish	PC 437	437
sv_SE.850	Swedish	PC 850	850

For further information listing inter-platform support for these locales, see the *MQSeries Distributed Queuing Guide.*

Appendix F. Stopping and removing queue managers manually

If the normal methods for stopping and removing queue managers fail, you can resort to the more drastic methods described here.

Stopping a queue manager manually

The normal method of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, use the following procedure to stop it manually:

1. Find the process IDs of the queue manager programs that are still running. Use the MQSeries for IRIX ps command. For example, if the queue manager is called QMNAME, the following command can be used:

    ps -ef | grep QMNAME

2. End the queue manager processes which are still running. Use the MQSeries for IRIX kill command, together with the process IDs discovered in the previous step.

Note: Processes that fail to stop can be ended using `kill -9`.

   End the processes in the following order:

> amqhasmx - logger
>
> amqharmx - log formatter, used only if the queue manager has linear logging selected
>
> amqzllp0 - checkpoint processor
>
> amqzlaa0 - queue manager agent(s)
>
> amqzxma0 - processing controller

Note: Manual ending of the queue manager may result in FFST's being taken, and the production of FDC files in `/var/mqm/errors`. This should not be regarded as a defect in the queue manager.

The queue manager should restart normally, even after being ended by use of the preceding method.

If you want to delete the queue manager after stopping it manually, use the **dltmqm** command as normal. If, for some reason, this command fails to delete the queue manager, the manual process detailed in <u>"Removing queue managers manually"</u> can be used.

Removing queue managers manually

You should note that manual removal of a queue manager is potentially very disruptive, particularly if multiple queue managers are being used on a single system. The reason is, that complete removal of a queue manager requires deletion of files, shared memory and semaphores. As it is impossible to identify which shared memory and semaphores belong to a particular queue manager, it is necessary to stop all running queue managers.

If you need to delete a queue manager manually, use the following procedure:

1. Stop all queue managers running on the machine from which you need to remove the queue manager.

2. Locate the queue manager directory from the configuration file `/var/mqm/mqs.ini` and look for the QueueManager stanza naming the queue manager to be deleted.

   Its Prefix and Directory attributes identify the queue manager directory. For a Prefix attribute of <Prefix> and a Directory attribute of <Directory>, the full path to the queue manager directory is:

   <Prefix>/qmgrs/<Directory>

3. Locate the queue manager log directory from the qm.ini configuration file in the queue manager directory. The LogPath attribute of the Log stanza identifies this directory.

4. Delete the queue manager directory, all subdirectories and files.

5. Delete the queue manager log directory, all subdirectories and files.

6. Remove the queue manager's `QueueManager` stanza from the `/var/mqm/mqs.ini` configuration file.

7. If the queue manager being deleted is also the default queue manager, remove the `DefaultQueueManager` stanza from the `/var/mqm/mqs.ini` configuration file.

8. Either, remove all shared memory and semaphores owned by the `mqm` user ID and `mqm` group, or restart the machine. Shared resources can be identified using the `ipcs` command and can be removed with the `ipcrm` command.

Appendix G. Messages

This appendix describes the format of the messages issued by MQSeries and how they are documented.

Message format

The format of the MQSeries messages is as follows:

- The message identifier, where the identifier has two components:

    1. The characters "AMQ", which identify the message as originating from MQSeries

    2. A four-digit decimal code.

- Text of the message

Structure of messages

This section describes the structure of MQSeries messages.

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as "message variables". The message variables are indicated by the use of the '&' symbol.

Where there is more than one variable in a message, a number is added to the '&' symbol.

Note: You should always look at the extended help for a message before carrying out any other action, because, in certain cases, the variables are only displayed in the extended help.

Message information

For each message, this information is provided:

Explanation:

Why the message was issued.

User action:

Instructions to the user.

Note: The message file may contain the explanation of the message, in addition to the message itself.

MQSeries messages

MQSeries messages are numbered 5000 through 9999, and they are listed in this book in numeric order. However, not all numbers have been used, and therefore, the list is not continuous.

Message groups

MQSeries messages are grouped according to the part of MQSeries from which they originate:

5000 through 5999

Installable services - see topic "Installable services messages".

6000 through 6999

Common services - see topic "Common services messages".

7000 through 7999

The MQSeries product - see topic "MQSeries product messages".

8000 through 8999

Administering MQSeries - see topic "Administration messages".

9000 through 9999

Remote - see topic "Remote messages".

Installable services messages

AMQ5006 Unexpected error: rc = &1
Explanation: An unexpected error occurred in an internal function of the product.
User action: Save the generated output files and contact your Willow Technology support center.

AMQ5501 There was not enough storage to satisfy the request
Explanation: An internal function of the product attempted to obtain storage, but there was none available.
User action: Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your Willow Technology support center.

AMQ5511 Installable service component '&3' returned '&4'.
Explanation: The internal function, that adds a component to a service, called the component initialization process. This process returned an error.
User action: Check the component was installed correctly. If it was, and the component was supplied by Willow Technology, then save the generated output files and contact your Willow Technology support center. If the component was not supplied by Willow Technology, save the generated output files and follow the support procedure for that component.

AMQ5512 Installable service component '&3' returned '&4' for queue manager name = '&5'.
Explanation: An installable service component returned an unexpected return code.
User action: Check the component was installed correctly. If it was, and the component was supplied by Willow Technology, then save the generated output files and contact your Willow Technology support center. If the component was not supplied by Willow Technology, save the generated output files and follow the support procedure for that component.

AMQ5513 '&3' returned &1.
Explanation: An unexpected error occurred.
User action: Save the generated output files and contact your Willow Technology support center.

AMQ5600 Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

AMQ5603 Usage: dltmqm [-z] QMgrName

AMQ5604 Usage: dspmqaut [-m QMgrName] [-n ObjName] -t ObjType [-p Principal | -g Group] [-s ServiceName]

AMQ5605 Usage: endmqm [-z] [-c | -i | -p] QMgrName

AMQ5606 Usage: setmqaut -m QMgrName [-n ObjName] -t ObjType [-p Principal | -g Group] [-s ServiceName] Authorizations

AMQ5607 Usage: strmqm [-z] [QMgrName]

AMQ5608 Usage: dspmqtrn QMgrName

AMQ5609 Usage: rsvmqtrn -m QMgrName (-c | -b) Transaction,Number
Explanation:

User action: None.

AMQ5610 Usage: strmqtrc [-m QMgrName] [-e] [-t TraceType]

AMQ5611 Usage: endmqtrc [-m QMgrName] [-e] [-a]

AMQ5612 Usage: dspmqtrc [-t TemplateFile] InputFileName

Common services messages

AMQ6004 An error occurred during MQSeries initialization or ending.
Explanation: An error was detected during initialization or ending of MQSeries. The MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6025 Program not found.
Explanation: MQSeries is unable to start program &3 because it was not found.
User action: Check the program name is correctly specified and rerun the program.

AMQ6026 A resource shortage prevented the creation of an MQSeries process.
Explanation: An attempt to create an MQSeries process was rejected by the operating system due to a process limit (either the number of processes for each user or the total number of processes running system wide), or because the system does not have the resources necessary to create another process.
User action: Investigate if a process limit is preventing the creation of the process and if so why the system is constrained in this way. Consider raising this limit or reducing the workload on the system.

AMQ6035 MQSeries failed, no storage available.
Explanation: An internal function of the product attempted to obtain storage, but there was none available.
User action: Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your Willow Technology support center.

AMQ6037 MQSeries was unable to obtain enough storage.
Explanation: The product is unable to obtain enough storage. The product's error recording routine may have been called.
User action: Stop the product and restart it. If this does not resolve the problem see if a problem has been recorded. If a problem has been recorded, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6047 Conversion not supported.
Explanation: MQSeries is unable to convert string data tagged in CCSID &1 to data in CCSID &2.
User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6048 DBCS error
Explanation: MQSeries is unable to convert string data due to a DBCS error. Conversion is from CCSID &1 to CCSID &2.
User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6049 DBCS only string not valid.
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2. Message descriptor data must be in single byte form. CCSID &2 is a DBCS only CCSID.

User action: Check the CCSID of your job or system and change it to one supporting SBCS or mixed character sets. Refer to the appropriate National Language Support publications for character sets and CCSIDs supported.

AMQ6050 CCSID error.
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.
User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6051 Conversion length error.
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, due to an input length error.
User action:

AMQ6052 Conversion length error.
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.
User action:

AMQ6053 CCSID error
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.
User action: One of the CCSIDs is not supported by the system. Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6064 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6090 MQSeries was unable to display an error message.
Explanation: MQSeries has attempted to display the message associated with return code &6. The return code indicates that there is no message text associated with the message. Associated with the request are inserts &1 : &2 : &3 : &4 : &5.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6091 An internal MQSeries error has occurred.
Explanation: Private memory has detected an error, and is abending due to &3. The error data is &1.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6100 An internal MQSeries error has occurred.
Explanation: MQSeries has detected an error, and is abending due to &3. The error data is &1.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6107 CCSID not supported.
Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, because one of the CCSIDs is not recognized.
User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6115 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6118 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6119 An internal MQSeries error has occurred.
Explanation: MQSeries detected an unexpected error when calling the operating system. The MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6120 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6121 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: MQSeries has detected a parameter count of &1 that is not valid. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6122 An internal MQSeries error has occurred.
Explanation: An error has been detected, and the MQSeries error recording routine has been called.
User action: MQSeries has detected parameter &1 that is not valid, having value &2&3. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6125 An internal MQSeries error has occurred.
Explanation: An internal error has occurred with identifier &1. This message is issued in association with other messages.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6148 An internal MQSeries error has occurred.
Explanation: MQSeries has detected an error, and is abending due to &3. The error data is &1.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ6172 No codeset found for current locale.
Explanation: No codeset could be determined for the current locale. Check that the locale in use is supported.
User action: None.

AMQ6173 No CCSID found for codeset &3.
Explanation: Codeset &3. has no supported CCSID. Check that the locale in use is supported. CCSIDs can be added by updating the file /var/mqm/conv/table/ccsid.tbl.
User action: None.

AMQ6708 A disk full condition was encountered when formatting a new log file in location &3.
Explanation: The queue manager attempted to format a new log file in directory &3. The drive or file system containing this directory did not have sufficient free space to contain the new log file.
User action: Increase the amount of space available for log files and retry the request.

AMQ6710 Queue manager unable to access directory &3.
Explanation: The queue manager was unable to access directory &3 for the log. This could be because the directory does not exist, or because the queue manager does not have sufficient authority.
User action: Ensure that the directory exists and that the queue manager has authority to read and write to it. Ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ6767 Log file &3 could not be opened for use.
Explanation: Log file &3 could not be opened for use. Possible reasons include the file being missing, the queue manager being denied permission to open the file or the contents of the file being incorrect.
User action: If the log file was required to start the queue manager, ensure that the log file exists and that the queue manager is able to read from and write to it. If the log file was required to recreate an object from its media image and you do not have a copy of the required log file, delete the object instead of recreating it.

MQSeries product messages

AMQ7001 The location specified for creation of the queue manager is not valid.
Explanation: The directory under which queue managers are to be created is not valid. It may not exist, or there may be a problem with authorization.
User action: The location is specified in the machine-wide ini file. Correct the file and submit the request again.

AMQ7002 An error occurred manipulating a file.
Explanation: An internal error occurred while trying to create or delete a queue manager file. It is likely that the error was caused by there being insufficient space on a disk, or by problems with authorization to the underlying filesystem.
User action: Identify the file that caused the error, using problem determination techniques. Correct the error in the filesystem and submit the request again.

AMQ7005 The queue manager is running.
Explanation: You tried to perform an action that requires the queue manager stopped, however, it is currently running. You probably tried to delete or start a queue manager that is currently running.
User action: If the queue manager should be stopped, stop the queue manager and submit the failed command again.

AMQ7006 Missing attribute &5 on stanza starting on line &1 of ini file &3.
Explanation: The &4 stanza starting on line &1 of configuration file &3 is missing the required &5 attribute.
User action: Check the contents of the file and retry the operation.

AMQ7008 The queue manager already exists.
Explanation: You tried to create a queue manager that already exists.
User action: If you specified the wrong queue manager name, correct the name and submit the request again.

AMQ7010 The queue manager does not exist.
Explanation: You tried to perform an action against a queue manager that does not exist. You may have specified the wrong queue manager name.
User action: If you specified the wrong name, correct it and submit the command again. If the queue manager should exist, create it, and then submit the command again.

AMQ7012 The specified trigger interval is not valid.
Explanation: You specified a value for the trigger interval that is not valid. The value must be not less than zero and not greater than 999 999 999.
User action: Correct the value and resubmit the request.

AMQ7013 There is an error in the name of the specified dead letter queue.
Explanation: You specified a name for the dead letter queue that is not valid.
User action: Correct the name and resubmit the request.

AMQ7014 There is an error in the name of the specified default transmission queue.
Explanation: You specified a name for the default transmission queue that is not valid.
User action: Correct the name and submit the command again.

AMQ7015 There is an error in the maximum number of open object handles specified.
Explanation: You specified a value for the maximum number of open object handles to be allowed that is not valid. The value must be not less than zero and not greater than 999 999 999.
User action: Correct the value and submit the command again.

AMQ7016 There is an error in the maximum number of uncommitted messages specified.
Explanation: You specified a value for the maximum number of uncommitted messages to be allowed that is not valid. The value must be not less than 1 and not greater than 999 999 999.
User action: Correct the value and submit the command again.

AMQ7017 Log not available.
Explanation: The queue manager was unable to use the log. This could be due to a log file being missing or damaged, or the log path to the queue manager being inaccessible.
User action: Ensure that the LogPath attribute in the queue manager configuration file is correct. If a log file is missing or otherwise unusable, restore a backup copy of the file, or the entire queue manager.

AMQ7018 The queue manager has stopped

AMQ7019 An error occurred while creating the directory structure for the new queue manager.
Explanation: During creation of the queue manager an error occurred while trying to create a file or directory.
User action: Identify why the queue manager files cannot be created. It is probable that there is insufficient space on the specified disk, or that there is a problem with access control. Correct the problem and submit the command again.

AMQ7021 An error occurred while deleting the directory structure for the queue manager.
Explanation: While deleting the queue manager, an error occurred deleting a file or directory. The queue manager may not have been completely deleted.
User action: Follow problem determination procedures to identify the file or directory and to complete deletion of the queue manager.

AMQ7024 Arguments supplied to a command are not valid.
Explanation: You supplied arguments to a command that it could not interpret. It is probable that you specified a flag not accepted by the command, or that you included extra flags.
User action: Correct the command and submit it again.

AMQ7025 Error in the supplied command description.
Explanation: The descriptive text you supplied on the command was in error.
User action: Correct the descriptive text and submit the command again.

AMQ7026 A principal or group name was invalid.
Explanation: You specified the name of a principal or group which does not exist.
User action: Correct the name and resubmit the request.

AMQ7028 The queue manager is not available for use.
Explanation: You have requested an action that requires the queue manager running, however, the queue manager is not currently running.
User action: Start the required queue manager and submit the command again.

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.
User action: None.

AMQ7031 The queue manager is stopping.
Explanation: You issued a command that requires the queue manager running, however, it is currently in the process of stopping. The command cannot be run.
User action: None

AMQ7041 Object already exists.
Explanation: A Define Object operation was performed, but the name selected for the object is already in use by an object that is unknown to MQSeries. The object name selected by MQSeries was &3, in directory &4, of object type &5.
User action: Remove the conflicting object from the MQSeries system, then try the operation again.

AMQ7042 Media image not available for object &3 of type &4.
Explanation: The media image for object &3, type &4, is not available for media recovery. A log file containing part of the media image cannot be accessed.
User action: A previous message indicates which log file could not be accessed. Restore a copy of the log file and all subsequent log files from backup. If this is not possible, you must delete the object instead.

AMQ7044 Media recovery not allowed.
Explanation: Media recovery is not possible on a queue manager using a circular log. Damaged objects must be deleted on such a queue manager.
User action: None.

AMQ7047 An unexpected error was encountered by a command.
Explanation: An internal error occurred during the processing of a command.
User action: Follow problem determination procedures to identify the cause of the error.

AMQ7048 The queue manager name is either not valid or not known
Explanation: Either the specified queue manager name does not conform to the rules required by MQSeries or the queue manager does not exist. The rules for naming MQSeries objects are detailed in the MQSeries Command Reference.
User action: Correct the name and submit the command again.

AMQ7053 The transaction has been committed.
Explanation: The prepared transaction has been committed.
User action: None.

AMQ7054 The transaction has been backed out.
Explanation: The prepared transaction has been backed out.
User action: None.

AMQ7055 The transaction number is not recognized.

Explanation: The number of the transaction you supplied was not recognized as belonging to an in-doubt transaction.
User action: Ensure that you entered a valid transaction number. It is possible that the transaction number you entered corresponds to a transaction which was committed or backed out before you issued the command to resolve it.

AMQ7056 Transaction number &1,&2.
Explanation: This message is used to report the number of an in-doubt transaction.
User action: None.

AMQ7064 Log path not valid or inaccessible.
Explanation: The supplied log path could not be used by the queue manager. Possible reasons for this include the path not existing, the queue manager not being able to write to the path, or the path residing on a remote device.
User action: Ensure that the log path exists and that the queue manager has authority to read and write to it. If the queue manager already exists, ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ7065 Insufficient space on disk.
Explanation: The operation cannot be completed due to shortage of disk space.
User action: Either make more disk space available, or reduce the disk requirements of the command you issued.

AMQ7066 There are no prepared transactions.
Explanation: There are no prepared transactions to be resolved.
User action: None.

AMQ7068 Authority file contains an authority stanza that is not valid.
Explanation: A syntax error has been found in one of the files containing authorization information for the queue manager.
User action: Correct the contents of the incorrect authorization file by editing it.

AMQ7069 The queue manager was created successfully, but cannot be made the default.
Explanation: The queue manager was defined to be the default queue manager for the machine when it was created. However, although the queue manager has been created, an error occurred trying to make it the default. There may not be a default queue manager defined for the machine at present.
User action: There is probably a problem with the machine-wide ini file. Verify the existence of the file, its access permissions, and its contents. If its backup file exists, reconcile the contents of the two files and then delete the backup. Finally, either update the machine-wide ini file by hand to specify the desired default queue manager, or delete and recreate the queue manager.

AMQ7073 Log size not valid.
Explanation: Either the number of log files or the size of the log files was outside the accepted values.
User action: Make sure that the log parameters you enter lie within the valid range.

AMQ7074 Unknown stanza key &4 on line &1 of ini file &3.
Explanation: Line &1 of the configuration file &3 contained a stanza called &3. This stanza is not recognized.

User action: Check the contents of the file and retry the operation.

AMQ7075 Unknown attribute &4 on line &1 of ini file &3.
Explanation: Line &1 of the configuration file &3 contained an attribute called &4 that is not valid. This attribute is not recognized in this context.
User action: Check the contents of the file and retry the operation.

AMQ7076 Value &5 not valid for attribute &4 on line &1 of ini file &3
Explanation: Line &1 of the configuration file &3 contained value &5 that is not valid for the attribute &4.
User action: Check the contents of the file and retry the operation.

AMQ7077 You are not authorized to perform the requested operation.
Explanation: You tried to issue a command for the queue manager. You are not authorized to perform the command.
User action: Contact your system administrator to perform the command for you. Alternatively, request authority to perform the command from your system administrator.

AMQ7080 No objects processed.
Explanation: No objects were processed, either because no objects matched the criteria given, or because the objects found did not require processing.
User action: None.

AMQ7081 Object &3, type &4 recreated.
Explanation: The object &3, type &4 was recreated from its media image.
User action: None.

AMQ7082 Object &3, type &4 is not damaged.
Explanation: Object &3, type &4 cannot be recreated since it is not damaged.
User action: None

AMQ7083 A resource problem was encountered by a command.
Explanation: The command failed due to a resource problem. Possible causes include the log being full or the command running out of memory.
User action: Look at the previous messages to diagnose the problem. Rectify the problem and retry the operation.

AMQ7084 Object &3, type &4 damaged.
Explanation: The object &3, type &4 was damaged. The object must be deleted or, if the queue manager supports media recovery, recreated from its media image.
User action: Delete the object or recreate it from its media image.

AMQ7085 Object &3, type &4 not found.
Explanation: Object &3, type &4 cannot be found.
User action: None.

AMQ7086 Media image for object &3, type &4 recorded.
Explanation: The media image for object &3, type &4 has been recorded.
User action: None.

AMQ7087 Object &3, type &4 is a temporary object
Explanation: Object &3, type &4 is a temporary object. Media recovery operations are not permitted on temporary objects.
User action: None.

AMQ7088 Object &3, type &4 in use.
Explanation: Object &3, type &4 is in use. Either an application has it open or, if it is a local queue, there are uncommitted messages on it.
User action: Ensure that the object is not opened by any applications, and that there are no uncommitted messages on the object, if it is a local queue. Then, retry the operation.

AMQ7089 Media recovery already in progress.
Explanation: Another media recovery operation is already in progress. Only one media recovery operation is permitted at a time.
User action: Wait for the existing media recovery operation to complete and retry the operation.

AMQ7090 The queue manager CCSID is not valid.
Explanation: The CCSID to be used by the QMGR is not valid, probably because it is a DBCS CCSID.
User action: None.

AMQ7091 You are performing authorization for the queue manager, but you specified an object name.
Explanation: Modification of authorizations for a queue manager can be performed only from that queue manager. You must not specify an object name.
User action: Correct the command and submit it again.

AMQ7092 An object name is required but you did not specify one.
Explanation: The command needs the name of an object, but you did not specify one.
User action: Correct the command and submit it again.

AMQ7093 An object type is required but you did not specify one.
Explanation: The command needs the type of the object, but you did not specify one.
User action: Correct the command and submit it again.

AMQ7094 You specified an object type that is not valid, or more than one object type.
Explanation: Either the type of object you specified was not valid, or you specified multiple object types on a command which supports only one.
User action: Correct the command and submit it again.

AMQ7095 An entity name is required but you did not specify one.
Explanation: The command needs one or more entity names, but you did not specify any. Entities can be principals or groups.
User action: Correct the command and submit it again.

AMQ7096 An authorization specification is required but you did not provide one.
Explanation: The command sets the authorizations on MQSeries objects. However you did not specify which authorizations are to be set.
User action: Correct the command and submit it again.

AMQ7097 You gave an authorization specification that is not valid.
Explanation: The authorization specification you provided to the command contained one or more items that could not be interpreted.
User action: Correct the command and submit it again.

AMQ7098 The command accepts only one entity name. You specified more than one.
Explanation: The command can accept only one principal or group name. You specified more than one.
User action: Correct the command and submit it again.

AMQ7099 Entity &3 has the following authorizations for object &4:
Explanation: Informational message. The list of authorizations follows.
User action: None.

AMQ7305 Trigger message could not be put on an initiation queue.
Explanation: The attempt to put a trigger message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the dead-letter queue.
User action: Ensure that the initiation queue is available, and operational.

AMQ7306 The dead-letter queue must be a local queue.
Explanation: An undelivered message has not been put on the dead-letter queue &4 on queue manager &5, because the queue is not a local queue. The message will be discarded.
User action: Inform your system administrator.

AMQ7307 A message could not be put on the dead-letter queue.
Explanation: The attempt to put a message on the undelivered-message queue &4 on queue manager &5 failed with reason code &1. The message will be discarded.
User action: Ensure that the undelivered-message queue is available, and operational.

AMQ7308 Trigger condition &1 was not satisfied.
Explanation: At least one of the conditions required for generating a trigger message was not satisfied, so a trigger message was not generated. If you were expecting a trigger message, consult the MQSeries Application Programming Guide for a list of the conditions required. (Note that arranging for condition &1 to be satisfied might not be sufficient because the conditions are checked in an arbitrary order, and checking stops when the first unsatisfied condition is discovered.)
User action: If a trigger message is required, ensure that all the conditions for generating one are satisfied.

AMQ7310 Report message could not be put on a reply-to queue.
Explanation: The attempt to put a report message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the undelivered-message queue.
User action: Ensure that the reply-to queue is available, and operational.

AMQ7463 The log for queue manager &3 is full.
Explanation: This message is issued when an attempt to write a log record is rejected because the log is full. The queue manager will attempt to resolve the problem.
User action: This situation may be encountered during a period of unusually high message traffic. However, if you persistently fill the log, you may have to consider enlarging the size of the log. You can either increase the number of log files by changing the values in the queue manager configuration file. You will then have to stop and restart the queue manager. Alternatively, if you

need to make the log files themselves bigger, you will have to delete and recreate the queue manager.

AMQ7464 The log for queue manager &3 is no longer full.
Explanation: This message is issued when a log was previously full, but an attempt to write a log record has now been accepted. The log full situation has been resolved.
User action: None

AMQ7465 The log for queue manager &3 is full. This is due to the presence of a long-running transaction.
Explanation: This message is issued when an attempt made to resolve a log full situation fails, because the space is occupied by a long-running transaction.
User action: Try to ensure that the duration of your transactions is not excessive. Commit or roll back any old transactions to release log space for further log records.

AMQ7466 The log for queue manager &3 is too small to support the current data rate.
Explanation: This message is issued when the monitoring tasks maintaining the log cannot keep up with the current rate of data being written.
User action: The number of primary log files configured should be increased to prevent possible log full situations.

AMQ7467 The oldest log file required to start queue manager &3 is &4.
Explanation: The log file &4 contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.
User action: You can move log files older than &4 to an archive medium to release space in the log directory. If you move any of the log files required to recreate objects from their media images, you will have to restore them to recreate the objects.

AMQ7468 The oldest log file required to perform media recovery of queue manager &3 is &4.
Explanation: The log file &4 contains the oldest log record required to recreate any of the objects from their media images. Any log files prior to this will not be accessed by media recovery operations.
User action: You can move log files older than &4 to an archive medium to release space in the log directory.

AMQ7469 Transactions rolled back to release log space.
Explanation: The log space for the queue manager is becoming full. One or more long-running transactions have been rolled back to release log space so that the queue manager can continue to process requests.
User action: Try to ensure that the duration of your transactions is not excessive. You may consider increasing the size of the log to allow transactions to last longer before the log starts to become full.

AMQ7472 Object &3, type &4 damaged.
Explanation: Object &3, type &4 has been marked as damaged. This indicates that the queue manager was either unable to access the object in the file system, or that some kind of inconsistency with the data in the object was detected.
User action: If a damaged object is detected, the action performed depends on whether the queue manager supports media recovery and when the damage was detected. If the queue manager does not support media recovery, you must delete the object as no recovery is possible. If the queue

manager does support media recovery and the damage is detected during the processing performed when the queue manager is being started, the queue manager will automatically initiate media recovery of the object. If the queue manager supports media recovery and the damage is detected once the queue manager has started, it may be recovered from a media image using the rcrmqobj command or it may be deleted.

AMQ7901 The data-conversion exit &3 has not loaded.
Explanation: The data-conversion exit program, &3, failed to load. The internal function gave exception &4.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ7902 The data conversion exit &3 was not loaded. The operating system call &4 returned &1.
Explanation:
User action: Specify REPLACE to over-write the existing file, or choose a different output file name.

AMQ7903 The data-conversion exit &3 cannot be found.
Explanation: Message data conversion has been requested for an MQSeries message with a user-defined format, but the necessary data-conversion exit program, &3, cannot be found. The internal function gave exception &4.
User action: Check that the necessary data-conversion exit &3 exists.

AMQ7904 The data conversion exit &3 cannot be found, or loaded.
Explanation: Message data conversion was requested for an MQSeries message with a user-defined format, but the necessary data conversion exit program, &3, was not found, or loaded. The &4 function call gave a return code of &1.
User action: Check that the necessary data conversion exit routine exists one of the standard directories for dynamically loaded modules. If necessary, inspect the generated output to examine the message descriptor (MQMD structure) of the MQSeries message for which conversion was requested. This may help you to determine where the message originated.

AMQ7905 Unexpected exception &4 in data-conversion exit.
Explanation: The data-conversion exit program, &3, ended with an unexpected exception &4. The message has not been converted.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ7907 Unexpected exception in data-conversion exit.
Explanation: The data-conversion exit routine, &3, ended with an unexpected exception. The message has not been converted.
User action: Correct the error in the data-conversion exit routine.

AMQ7921 An internal MQSeries error occurred.
Explanation: The MQDXP structure passed to the Internal Formats Conversion routine contains an incorrect eyecatcher field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ7922 A PCF message is incomplete.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the message is only &1 bytes long and does not contain a PCF header. The message has either been truncated, or it contains data that is not valid.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7923 A message had an unrecognized integer encoding.

Explanation: Message data conversion cannot convert a message because the integer encoding value of the message, &1, was not recognized.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7924 Bad length in the PCF header (length = &1).

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the PCF header structure contains an incorrect length field. Either the message has been truncated, or it contains data that is not valid.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7925 Message version &1 is not supported.

Explanation: Message data conversion cannot convert a message because the Version field of the message contains an incorrect value.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7926 A PCF message has an incorrect parameter count value &1.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the parameter count field of the PCF header is incorrect.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7927 Bad type in PCF structure number &1 (type = &2).

Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect type field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7928 Bad length in PCF structure number &1 (length = &2).
Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect length field.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7929 A PCF structure is incomplete.
Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number &1, of Type value &2, within the message is incomplete. The message has either been truncated, or it contains data that is not valid.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7930 Bad CCSID in PCF structure number &1 (CCSID = &2).
Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contains an incorrect CCSID.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7931 Bad length in PCF structure number &1 (length = &2).
Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because one of the structures of the message contains an incorrect length field.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7932 Bad count in PCF structure number &1 (count = &2).
Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because a StringList structure of the message contains an incorrect count field.
User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message, and to see how data that is not valid became included in the message.

AMQ7933 Bad string length in PCF structure.
Explanation: Message data conversion cannot convert a message in Programmable Command

Format (PCF) because structure number &1 of the message contains an incorrect string length value &2.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7934 Wrong combination of MQCCSI_DEFAULT with MQCCSI_EMBEDDED.

Explanation: Message data conversion could not convert a message in Programmable Command Format (PCF) because structure &1 of the message contained a CodedCharSetId field of MQCCSI_DEFAULT while the message itself had a CodedCharSetId of MQCCSI_EMBEDDED. This is an incorrect combination.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7935 Bad CCSID in message header (CCSID = &1).

Explanation: Message data conversion could not convert a message because the Message Descriptor of the message contained an incorrect CodedCharSetId field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7936 The file &3 already exists.

Explanation: The output file already exists, but REPLACE has not been specified.

User action: Specify REPLACE to over-write the existing file, or select a different output file name.

AMQ7943 Usage: crtmqcvx SourceFile TargetFile

AMQ7953 One structure has been parsed.

Explanation: The crtmqcvx command has parsed one structure.

User action: None.

AMQ7954 &1 structures have been parsed.

Explanation: The crtmqcvx command has parsed %1 structures.

User action: None.

AMQ7955 Unexpected field: &1.

Explanation: The field within the structure is of a type that is not recognized.

User action: Correct the field and retry the command.

AMQ7956 Bad array dimension.

Explanation: An array field of the structure has an incorrect dimension value.

User action: Correct the field and retry the command.

AMQ7957 Warning at line &1.

Explanation: The structure contains another field after a variable length field.
User action: Correct the structure and retry the command.

AMQ7958 Error at line &1 in field &3.
Explanation: Field name '&3' is a field of type 'float'. Fields of type float are not supported by this command.
User action: Either correct the structure to eliminate fields of type float, or write your own routine to support conversion of these fields.

AMQ7959 Error at line &1 in field &3.
Explanation: Field name '&3' is a field of type 'double'. Fields of type double are not supported by this command.
User action: Either correct the structure to eliminate fields of type double, or write your own routine to support conversion of these fields.

AMQ7960 Error at line &1 in field &3.
Explanation: Field name '&3' is a 'pointer' field. Fields of type pointer are not supported by this command.
User action: Either correct the structure to eliminate fields of type pointer, or write your own routine to support conversion of these fields.

AMQ7961 Error at line &1 in field &3.
Explanation: Field name '&3' is a 'bit' field. Bit fields are not supported by this command.
User action: Either correct the structure to eliminate bit fields, or write your own routine to support conversion of these fields.

AMQ7962 No input file specified.
Explanation: This command requires that an input file is specified.
User action: Specify the name of the input file and retry the command.

AMQ7963 No output file specified.
Explanation: This command requires that an output file name is specified.
User action: Specify the name of the output file and retry the command.

AMQ7964 Unexpected option &3.
Explanation: The option specified is not valid for this command.
User action: Retry the command with a valid option.

AMQ7965 Incorrect number of arguments.
Explanation: The command was passed an incorrect number of arguments.
User action: Retry the command, passing it the correct number of arguments.

AMQ7968 Cannot open file '&3'.
Explanation: You cannot open the file &3.
User action: Check that you have the correct authorization to the file and retry the command.

AMQ7969 Syntax error.
Explanation: This line of the input file contains a language syntax error.
User action: Correct the syntax error and retry the command.

AMQ7970 Syntax error on line &1.
Explanation: This message identifies where, in the input file, a previously reported error was detected.
User action: Correct the error and retry the command.

Administration messages

AMQ8001 MQSeries queue manager created.
Explanation: MQSeries queue manager &5 created.
User action: None.

AMQ8002 MQSeries queue manager deleted.
Explanation: MQSeries queue manager &5 deleted.
User action: None.

AMQ8003 MQSeries queue manager started.
Explanation: MQSeries queue manager &5 started.
User action: None.

AMQ8004 MQSeries queue manager ended.
Explanation: MQSeries queue manager &5 ended.
User action: None.

AMQ8005 MQSeries queue manager changed.
Explanation: MQSeries queue manager &5 changed.
User action: None.

AMQ8006 MQSeries queue created.
Explanation: MQSeries queue &5 created.
User action: None.

AMQ8007 MQSeries queue deleted.
Explanation: MQSeries queue &5 deleted.
User action: None.

AMQ8008 MQSeries queue changed.
Explanation: MQSeries queue &5 changed.
User action: None.

AMQ8010 MQSeries process created.
Explanation: MQSeries process &5 created.
User action: None.

AMQ8011 MQSeries process deleted.
Explanation: MQSeries process &5 deleted.
User action: None.

AMQ8012 MQSeries process changed.
Explanation: MQSeries process &5 changed.
User action: None.

AMQ8013 MQM process copied.
Explanation: MQM process &5 created in library &3 by copying.
User action: None.

AMQ8014 MQSeries channel created.
Explanation: MQSeries channel &5 created.
User action: None.

AMQ8015 MQSeries channel deleted.
Explanation: MQSeries channel &5 deleted.
User action: None.

AMQ8016 MQSeries channel changed.
Explanation: MQSeries channel &5 changed.
User action: None.

AMQ8018 Start MQSeries channel accepted.
Explanation: MQSeries channel &5 is being started. The start channel function has been initiated. This involves a series of operations across the network before the channel is actually started. The channel status displays "BINDING" for a short period while communication protocols are negotiated with the channel with whom communication is being initiated.
User action: None.

AMQ8019 Stop MQSeries channel accepted.
Explanation: MQSeries channel &5 has been requested to stop.
User action: None.

AMQ8020 Ping MQSeries channel complete.
Explanation: Ping MQSeries channel &5 complete.
User action: None.

AMQ8021 MQSeries Listener program started.
Explanation: The MQSeries channel listener program has been started.
User action: None.

AMQ8022 MQSeries queue cleared.
Explanation: All messages on MQSeries queue &5 have been deleted.
User action: None.

AMQ8023 MQSeries channel reset.
Explanation: MQSeries channel &5 has been reset.
User action: None.

AMQ8024 MQSeries channel initiator started.
Explanation: The channel initiator for MQSeries queue &5 has been started.
User action: None.

AMQ8025 MQSeries channel resolved.
Explanation: In doubt messages for MQSeries channel &5 have been resolved.
User action: None.

AMQ8026 End MQSeries queue manager accepted.
Explanation: A controlled stop request has been initiated for MQSeries queue manager &5.
User action: None.

AMQ8027 MQSeries command server started.
Explanation: The MQSeries command server has been started.
User action: None.

AMQ8028 MQSeries command server ended.
Explanation: The MQSeries command server has been stopped.
User action: None.

AMQ8029 MQSeries authority granted.
Explanation: Authority for MQSeries object &5 granted.
User action: None.

AMQ8030 MQSeries authority revoked.
Explanation: Authority for MQSeries object &5 revoked.
User action: None.

AMQ8033 MQSeries object recreated.
Explanation: MQSeries object &5 has been recreated from image.
User action: None.

AMQ8034 MQSeries object image recorded.
Explanation: Image of MQSeries object &5 has been recorded.
User action: None.

AMQ8035 MQSeries Command Server Status . . : Running

AMQ8036 MQSeries command server status . . : Stopping

AMQ8037 MQSeries command server status . . : Starting

AMQ8038 MQSeries command server status . . : Running with queue disabled

AMQ8039 MQSeries command server status . . : Stopped

AMQ8040 MQSeries command server ending.

AMQ8041 The queue manager cannot be restarted because processes, that were previously connected, are still running.
Explanation: Processes, that were connected to the queue manager the last time it was running, are still active. The queue manager cannot be restarted.
User action: Stop the processes and try to start the queue manager.

AMQ8042 Process &1 is still running.

AMQ8043 Non runtime application attempted to connect to runtime only queue manager.
Explanation: A non runtime application attempted to connect to a queue manager on a node where support for non runtime applications has not been installed. The connect attempt will be rejected with a reason of MQRC_ENVIRONMENT_ERROR.
User action: If the node is intended to support only runtime applications then investigate why a non runtime application has attempted to connect to the queue manager. If the node is intended to support non runtime only applications then investigate if the base option has been installed. The

base option must be installed if non runtime applications are to run on this node.

AMQ8101 Unexpected error (&1).
Explanation: An unexpected reason code with hexadecimal value &4 was received from the MQSeries queue manager during command processing. (Note that hexadecimal values in the range X'07D1'-X'0BB7' correspond to MQI reason codes 2001-2999.) More information might be available in the log. If the reason code value indicates that the error was associated with a particular parameter, the parameter concerned is &2.
User action: Correct the error and then try the command again.

AMQ8102 MQSeries object name specified in &2 not valid.
Explanation: MQSeries object name &5 specified in &2 is not valid. The length of the name must not exceed 48 characters, or 20 characters if it is a channel name. The name should contain the following characters only: lowercase a-z, uppercase A-Z, numeric 0-9, period (.), forward slash (/), underscore (_) and percent sign (%).
User action: Change the length of the parameter value or change the parameter value to contain a valid combination of characters, then try the command again.

AMQ8103 Insufficient storage available.
Explanation: There was insufficient storage available to perform the requested operation.
User action: Free some storage and then try the command again.

AMQ8104 MQSeries directory &3 not found.
Explanation: Directory &3 was not found. This directory is created when MQSeries is installed successfully. Refer to the log for more information.
User action: Verify that installation of MQSeries was successful. Correct the error and then try the command again.

AMQ8105 Object error.
Explanation: An object error occurred. Refer to the log for more information.
User action: Correct the error and then try the command again.

AMQ8106 MQSeries queue manager being created.
Explanation: The MQSeries queue manager is being created.
User action: Wait for the creation process to complete and then try the command again.

AMQ8107 MQSeries queue manager running.
Explanation: The MQSeries queue manager is running.
User action: None.

AMQ8108 MQSeries queue manager ending.
Explanation: The MQSeries queue manager is ending.
User action: Wait for the MQSeries queue manager to end and then try the command again.

AMQ8109 MQSeries queue manager being deleted.
Explanation: The MQSeries queue manager is being deleted.
User action: Wait for the deletion process to complete.

AMQ8110 MQSeries queue manager already exists.
Explanation: MQSeries queue manager &5 already exists.

User action: None.

AMQ8117 MQSeries queue manager deletion incomplete.
Explanation: Deletion of MQSeries queue manager &5 was only partially successful. An object was not found, or could not be deleted. Refer to the log for more information.
User action: Delete any remaining MQSeries queue manager objects.

AMQ8118 MQSeries queue manager does not exist.
Explanation: MQSeries queue manager &5 does not exist.
User action: Create the message queue manager (crtmqm command) and then try the command again.

AMQ8135 Not authorized.
Explanation: You are not authorized to perform the requested operation for the MQSeries object &5 specified in &2. Either you are not authorized to perform the requested operation, or you are not authorized to the specified MQSeries object. For a copy command, you may not be authorized to the specified source MQSeries object, or, for a create command, you may not be authorized to the system default MQSeries object of the specified type.
User action: Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8137 MQSeries queue manager already starting.
Explanation: The strmqm command was unsuccessful because MQSeries queue manager &5 is already starting.
User action: Wait for the strmqm command to complete.

AMQ8138 The MQSeries queue has an incorrect type.
Explanation: The operation is not valid with MQSeries queue &5 because it is not a local queue.
User action: Change the QNAME parameter to specify an MQSeries queue of the correct type.

AMQ8139 Already connected.
Explanation: A connection to the MQSeries queue manager already exists.
User action: None.

AMQ8140 Resource timeout error.
Explanation: A timeout occurred in the communication between internal MQSeries queue manager components. This is most likely to occur when the system is heavily loaded.
User action: Wait until the system is less heavily loaded, then try the command again.

AMQ8141 MQSeries queue manager starting.
Explanation: MQSeries queue manager &5 is starting.
User action: Wait for the MQSeries queue manager startup process to complete and then try the command again.

AMQ8142 MQSeries queue manager stopped.
Explanation: MQSeries queue manager &5 is stopped.
User action: Use the strmqm command to start the MQSeries queue manager, and then try the command again.

AMQ8143 MQSeries queue not empty.

Explanation: MQSeries queue &5 specified in &2 is not empty or contains uncommitted updates.
User action: Commit or rollback any uncommitted updates. If the command is DELETE QLOCAL, use the CLEAR QLOCAL command to clear the messages from the MQSeries queue. Then try the command again.

AMQ8144 Log not available.
Explanation: The MQSeries logging resource is not available.
User action: Use the dltmqm command to delete the MQSeries queue manager and then the crtmqm command to create the MQSeries queue manager. Then try the command again.

AMQ8145 Connection broken.
Explanation: The connection to the MQSeries queue manager failed during command processing. This may be caused by an endmqm -i command being issued by another user, or by an MQSeries queue manager error.
User action: Use the strmqm command to start the message queue manager, wait until the message queue manager has started, and try the command again.

AMQ8146 MQSeries queue manager not available.
Explanation: The MQSeries queue manager is not available because it has been stopped or has not been created.
User action: Use the crtmqm command to create the message queue manager, or the strmqm command to start the message queue manager as necessary. Then try the command again.

AMQ8147 MQSeries object not found.
Explanation: If the command entered was Change, the MQSeries object &5 specified in &2 does not exist. If the command entered was Copy, the source MQSeries object does not exist. If the command entered was Create, the system default MQSeries object of the specified type does not exist.
User action: Correct the MQSeries object name and then try the command again or, if you are creating a new MQSeries queue or process object, either specify all parameters explicitly or ensure that the system default object of the required type exists. The system default queue names are SYSTEM.DEFAULT.LOCAL.QUEUE, SYSTEM.DEFAULT.ALIAS.QUEUE and SYSTEM.DEFAULT.REMOTE.QUEUE. The system default process name is SYSTEM.DEFAULT.PROCESS.

AMQ8148 MQSeries object in use.
Explanation: MQSeries object &5 specified in &2 is in use by an MQSeries application program.
User action: Wait until the MQSeries object is no longer in use and then try the command again, or specify FORCE to force the processing of the MQSeries ALTER command regardless of any application program affected by the change. If the object is the dead-letter queue and the open input count is nonzero, it may be in use by an MQSeries channel. If the object is another MQSeries queue object with a nonzero open output count, it may be in use by an MQSeries channel (of type RCVR or RQSTR). In either case, use the STOP CHANNEL and START CHANNEL commands to stop and restart the channel in order to solve the problem.

AMQ8149 MQSeries object damaged.
Explanation: The MQSeries object &5 specified in &2 is damaged.
User action: The MQSeries object contents are not valid. Issue the DISPLAY CHANNEL, DISPLAY QUEUE, or DISPLAY PROCESS command, as required, to determine the name of the

damaged object. Issue the DEFINE command, for the appropriate object type, to replace the damaged object, then try the command again.

AMQ8150 MQSeries object already exists.
Explanation: MQSeries object &5 specified for &2 could not be created because it already exists.
User action: Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries object. Then try the command again.

AMQ8151 MQSeries object has different type.
Explanation: The type specified for MQSeries object &5 is different from the type of the object being altered or defined.
User action: Use the correct MQSeries command for the object type, and then try the command again.

AMQ8152 Source MQSeries object has different type.
Explanation: The type of the source MQSeries object is different from that specified.
User action: Correct the name of the command, or source MQSeries object name, and then try the command again, or try the command using the REPLACE option.

AMQ8153 Insufficient disk space for the specified queue.
Explanation: The command failed because there was insufficient disk space available for the specified queue.
User action: Release some disk space and then try the command again.

AMQ8155 Connection limit exceeded.
Explanation: The queue manager connection limit has been exceeded.
User action: The maximum limit on the number of MQSeries application programs that may be connected to the MQSeries queue manager has been exceeded. Try the command later.

AMQ8156 MQSeries queue manager quiescing.
Explanation: The MQSeries queue manager is quiescing.
User action: The queue manager was stopping with -c specified for endmqm. Wait until the queue manager has been restarted and then try the command again.

AMQ8157 Security error.
Explanation: An error was reported by the security manager program.
User action: Inform your systems administrator, wait until the problem has been corrected, and then try the command again.

AMQ8159 MAXDEPTH not allowed with queue type *ALS or *RMT.
Explanation: The MAXDEPTH parameter may not be specified for an MQM queue of type *ALS or *RMT.
User action: Remove the MAXDEPTH parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8160 DFTSHARE not allowed with queue type *ALS or *RMT.
Explanation: The DFTSHARE parameter may not be specified for an MQM queue of type *ALS or *RMT.
User action: Remove the DFTSHARE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8172 Already disconnected.
Explanation: The MQI reason code of 2018 was returned from the MQSeries queue manager in response to an MQDISC request issued during command processing.
User action: None.

AMQ8173 No processes to display.
Explanation: There are no matching processes defined on this system.
User action: Using the DEFINE PROCESS command to create a process.

AMQ8174 No queues to display.
Explanation: There are no matching queues defined on this system.
User action: Using the appropriate command to define a queue of the type that you require, that is, DEFINE QALIAS, DEFINE QLOCAL, DEFINE QMODEL, or DEFINE QREMOTE.

AMQ8185 Operating system object already exists.
Explanation: The MQSeries object cannot be created because an object that is not known to MQSeries already exists in the MQSeries directory with the name that should be used for the new object. Refer to the log for previous messages.
User action: Remove the non-MQSeries object from the MQSeries library, and try the command again.

AMQ8186 Image not available for MQSeries object &5.
Explanation: MQSeries object &5 type &3 cannot be recreated because the image is not fully available in the logs that are currently online. Refer to earlier messages in the error log for information about the error logs that need to be brought online for this object to be recreated.
User action: Bring the relevant error logs online, and try the command again.

AMQ8187 MQSeries object &5 is currently open.
Explanation: MQSeries object &5, type &3, is currently in use, so the &1 command cannot be issued against it. If a generic list was presented to the command, the command is still issued against the other objects in the list.
User action: Wait until the object is no longer in use, and try the command again.

AMQ8188 Insufficient authorization to MQSeries object &5.
Explanation: You are not authorized to issue the &1 command against MQSeries object &5 type &3. If a generic list was presented to the command, the command is still issued against the other objects in the list.
User action: Obtain sufficient authorization for the object, and retry the command.

AMQ8189 MQSeries object &5 is damaged.
Explanation: MQSeries object &5 type &3 is damaged and the &1 command cannot be issued against it. If a generic list was presented to the command then the command is still issued against the other objects in the list.
User action: Issue the appropriate DEFINE command for the object, specifying REPLACE, and then try the command again.

AMQ8190 &1 succeeded on &2 objects and failed on &3 objects.
Explanation: An operation performed on a generic list of objects was not completely successful.
User action: Examine the log for details of the errors encountered, and take appropriate action.

AMQ8191 MQSeries command server is starting.
Explanation: The MQSeries command server is starting.
User action: Wait for the strmqcsv command to complete and then try the operation again.

AMQ8192 MQSeries command server already starting.
Explanation: The request to start the MQSeries command server was unsuccessful because the MQSeries command server is already starting.
User action: Wait for the strmqcsv command to complete.

AMQ8193 MQSeries command server is ending.
Explanation: The MQSeries command server is ending.
User action: Wait for the endmqcsv command to complete and then try the command again.

AMQ8194 MQSeries command server already ending.
Explanation: The end MQSeries command server request was unsuccessful because the MQSeries command server is already ending.
User action: Wait for the endmqcsv command to complete.

AMQ8195 MQSeries command server already running.
Explanation: The strmqcsv command was unsuccessful because the MQSeries command server is already running.
User action: None.

AMQ8196 MQSeries command server already stopped.
Explanation: The request to end the MQSeries command server was unsuccessful because the MQSeries command server is already stopped.
User action: None.

AMQ8197 Deleted MQSeries queue damaged.
Explanation: The deleted MQSeries queue &5 was damaged, and any messages it contained have been lost.
User action: None.

AMQ8226 MQSeries channel already exists.
Explanation: MQSeries channel &3 cannot be created because it already exists.
User action: Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries channel and then try the command again.

AMQ8227 Channel &3 not found.
Explanation: ALTER CHANNEL has been issued for a non-existent channel.
User action: Correct the MQSeries channel name and then try the command again.

AMQ8296 &4 MQSC commands completed successfully.
Explanation: The &1 command has completed successfully. The &4 MQSeries commands from &5 have been processed without error and a report written to the printer spool file.
User action: None.

AMQ8297 &4 MQSC commands verified successfully.
Explanation: The &1 command completed successfully. The &4 MQSeries commands from &5 have been verified and a report written to the printer spool file.

User action: None.

AMQ8298 Error report generated for MQSC command process.
Explanation: The &1 command attempted to process the sequence of MQSeries commands from &5 and encountered some errors, however, the operation may have partially completed. A report has been written to the printer spool file.
User action: Examine the spooled printer file for details of the errors encountered, correct the MQSC source file, and retry the operation.

AMQ8299 Cannot open &5 for MQSC process.
Explanation: The &1 command failed to open &5 for MQSeries command processing.
User action: Check that the intended file exists, and has been specified correctly. Correct the specification or create the object, and try the operation again.

AMQ8302 Internal failure initializing MQSeries services.
Explanation: An error occurred while attempting to initialize MQSeries services.
User action:

AMQ8303 Insufficient storage available to process request.
Explanation:
User action:

AMQ8304 Tracing cannot be started. Too many traces are already running.
Explanation:
User action: Stop one or more of the other traces and try the command again.

AMQ8305 Tracing cannot be started. Too many traces are already running.
Explanation:
User action: Stop one or more of the other traces and try the command again.

AMQ8401 &1 MQSC commands read.
Explanation: The MQSC script contains &1 commands.
User action: None.

AMQ8402 &1 commands have a syntax error.
Explanation: The MQSC script contains &1 commands having a syntax error.
User action: None.

AMQ8403 &1 commands cannot be processed.
Explanation: The MQSC script contains &1 commands that failed to process.
User action: None.

AMQ8404 Command failed.
Explanation: An MQSC command has been recognized, but cannot be processed.
User action: None.

AMQ8405 Syntax error detected at or near end of command segment below:-
Explanation: The MQSC script contains &1 commands having a syntax error.
User action: None.

AMQ8406 Unexpected 'end of input' in MQSC.

Explanation: An MQSC command contains a continuation character, but the 'end of input' has been reached without completing the command.
User action: None.

AMQ8407 Display Process details.
Explanation: The MQSC DISPLAY PROCESS command completed successfully, and details follow this message.
User action: None.

AMQ8408 Display Queue Manager details.
Explanation: The MQSC DISPLAY QMGR command completed successfully, and details follow this message.
User action: None.

AMQ8409 Display Queue details.
Explanation: The MQSC DISPLAY QUEUE command completed successfully, and details follow this message.
User action: None.

AMQ8410 Parser Error.
Explanation: The MQSC Parser has an internal error.
User action: None.

AMQ8411 Duplicate Keyword Error.
Explanation: A command in the MQSC script contains duplicate keywords.
User action: None.

AMQ8412 Numeric Range Error.
Explanation: The value assigned to an MQSC command keyword is out of the permitted range.
User action: None.

AMQ8413 String Length Error.
Explanation: A string assigned to an MQSC keyword is either NULL, or longer than the maximum permitted for that keyword.
User action: None.

AMQ8414 Display Channel details.
Explanation: The MQSC DISPLAY CHL command completed successfully, and details follow this message.
User action: None.

AMQ8415 MQSeries commands are active.
Explanation: The MQSC DISPLAY QMGR command completed successfully, and details follow this message.
User action: None.

AMQ8416 MQSC timed out waiting for a response from the command server.
Explanation: MQSC did not receive a response message from the remote command server in the time specified.
User action: None.

AMQ8417 Display Channel Status details.
Explanation: The MQSC DISPLAY CHANNEL STATUS command completed successfully, and details follow this message.
User action: None.

AMQ8418 &1 command responses received.
Explanation: Running in queued mode, &1 command responses were received from the remote command server.
User action: None.

AMQ8419 The Queue is already in the DCE cell.
Explanation: The Queue is already in the cell, that is, its SCOPE attribute is already CELL.
User action: None.

AMQ8421 A required keyword was not specified.
Explanation: A keyword required in this command was not specified.
User action: None.

AMQ8498 Starting MQSeries Commands.
Explanation: The MQSC script contains &1 commands.
User action: None.

AMQ8499 Usage: runmqsc [-e] [-v] [-w WaitTime] [-x] QMgrName
Explanation: None.
User action: None.

AMQ8500 MQSeries Display MQ Files
Explanation: Title for the dspmqfls command.
User action: None.

AMQ8501 Common services initialization failed with return code &1.
Explanation: A request by the command server to initialize common services failed with return code &1.
User action: None.

AMQ8502 Connect shared memory failed with return code &1.
Explanation: A request by the command server to connect shared memory failed with return code &1.
User action: None.

AMQ8503 Post event semaphore failed with return code &1.
Explanation: A request by the command server to post an event semaphore failed with return code &1.
User action: None.

AMQ8504 Command server MQINQ failed with reason code &1.
Explanation: An MQINQ request by the command server, for the MQSeries queue &3, failed with reason code &1.
User action: None.

AMQ8505 Reallocate memory failed with return code &1.
Explanation: A request by the command server to reallocate memory failed with return code &1.
User action: None.

AMQ8506 Command server MQGET failed with reason code &1.
Explanation: An MQGET request by the command server, for the MQSeries queue &3, failed with reason code &1.
User action: None.

AMQ8507 Command server MQPUT1 request for an undelivered message failed with reason code &1.
Explanation: An attempt by the command server to put a message to the dead-letter queue, using MQPUT1, failed with reason code &1. The MQDLH reason code was &2.
User action: None.

AMQ8508 Queue Manager Delete Object List failed with return code &1.
Explanation: A request by the command server to delete a queue manager object list failed with return code &1.
User action: None.

AMQ8509 Command server MQCLOSE reply-to queue failed with reason code &1.
Explanation: An MQCLOSE request by the command server for the reply-to queue failed with reason code &1.
User action: None.

AMQ8511 Usage: strmqcsv QMgrName
Explanation:
User action:

AMQ8512 Usage: endmqcsv [-c | -i] QMgrName

AMQ8513 Usage: dspmqcsv QMgrName

AMQ8514 No response received after &1 seconds.
Explanation: The command server has not reported the status of running, to the start request, before the timeout of &1 seconds was reached.
User action: None.

AMQ8601 MQSeries trigger monitor started.
Explanation: The MQSeries trigger monitor has been started.
User action: None.

AMQ8602 MQSeries trigger monitor ended.
Explanation: The MQSeries trigger monitor has ended.
User action: None.

AMQ8603 Usage: runmqtrm [-m QMgrName] [-q InitQ]

AMQ8604 Use of MQSeries trigger monitor not authorized.
Explanation: The MQSeries trigger monitor cannot be run due to lack of authority to the requested queue manager or initiation queue.

User action: Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8605 Queue manager not available to the MQSeries trigger monitor
Explanation: The queue manager specified for the trigger monitor does not exist, or is not active.
User action: Check that you named the correct queue manager. Ask your systems administrator to start it, if it is not active. Then try the command again.

AMQ8606 Insufficient storage available for the MQSeries trigger monitor.
Explanation: There was insufficient storage available for the MQSeries trigger monitor to run.
User action: Free some storage and then try the command again.

AMQ8607 MQSeries trigger monitor connection failed.
Explanation: The trigger monitor's connection to the requested queue manager failed because of MQI reason code &1 from MQCONN.
User action: Consult your systems administrator about the state of the queue manager.

AMQ8608 MQSeries trigger monitor connection broken.
Explanation: The connection to the queue manager failed while the trigger monitor was running. This may be caused by an endmqm command being issued by another user, or by an MQSeries queue manager error.
User action: Consult your systems administrator about the state of the queue manager.

AMQ8609 Initiation queue missing or wrong type
Explanation: The named initiation queue could not be found; or the queue type is not correct for an initiation queue.
User action: Check that the named queue exists, and is a local queue, or that the named queue is an alias for a local queue which exists.

AMQ8610 Initiation queue in use
Explanation: The MQSeries trigger monitor could not open the initiation queue because the queue is open for exclusive use by another application.
User action: Wait until the queue is no longer in use, and try the command again.

AMQ8611 Initiation queue could not be opened.
Explanation: The MQSeries trigger monitor could not open the initiation queue; reason code &1 was returned from MQOPEN.
User action: Consult your systems administrator.

AMQ8612 Waiting for a trigger message
Explanation: The MQSeries trigger monitor is waiting for a message to arrive on the initiation queue.
User action: None.

AMQ8613 Initiation queue changed or deleted
Explanation: The MQSeries trigger monitor is unable to continue because the initiation queue has been deleted or changed since it was opened.
User action: Retry the command.

AMQ8614 Initiation queue not enabled for input.

Explanation: The MQSeries trigger monitor cannot read from the initiation queue because input is not enabled.
User action: Ask your systems administrator to enable the queue for input.

AMQ8615 MQSeries trigger monitor failed to get message.
Explanation: The MQSeries trigger monitor failed because of MQI reason code &1 from MQGET.
User action: Consult your systems administrator.

AMQ8616 End of application trigger.
Explanation: The action to trigger an application has been completed.
User action: None.

AMQ8617 Not a valid trigger message.
Explanation: The MQSeries trigger monitor received a message that is not recognized as
User action: Consult your systems administrator.

AMQ8618 Error starting triggered application.
Explanation: An error was detected when trying to start the application identified in a trigger message.
User action: Check that the application the trigger monitor was trying to start is available.

AMQ8619 Application type &1 not supported.
Explanation: A trigger message was received which specifies application type &1; the trigger monitor does not support this type.
User action: Use an alternative trigger monitor for this initiation queue.

AMQ8620 Trigger message with warning &1
Explanation: The trigger monitor received a message with a warning. For example, it may have been truncated or it could not be converted to the trigger monitor's data representation. The reason code for the warning is &1.
User action: None.

AMQ8621 Usage: runmqtmc [-m QMgrName] [-q InitQ]

AMQ8701 Usage: rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]

AMQ8702 Usage: rcrmqobj [-z] [-m QMgrName] -t ObjType [GenericObjName]

AMQ8703 Usage: dspmqfls [-m QMgrName] [-t ObjType] GenericObjName

AMQ8708 Dead letter queue handler started to process INPUTQ(&3).
Explanation: The dead letter queue handler (runmqdlq) has been started and has parsed the input file without detecting any errors and is about to start processing the queue identified in the message.
User action: None.

AMQ8709 Dead letter queue handler ending.
Explanation: The dead letter queue handler (runmqdlq) is ending because the WAIT interval has expired and there are no messages on the dead letter queue, or because the queue manager is shutting down, or because the dead letter queue handler has detected an error. If the dead letter queue handler has detected an error, an earlier message will have identified the error.

User action: None.

AMQ8721 Dead letter queue message not prefixed by a valid MQDLH.

Explanation: The dead letter queue handler (runmqdlq) retrieved a message from the nominated dead letter queue, but the message was not prefixed by a recognizeable MQDLH. This typically occurs because an application is writing directly to the dead letter queue but is not prefixing messages with a valid MQDLH. The message is left on the dead letter queue and the dead letter queue handler continues to process the dead letter queue. Each time the dead letter queue handler repositions itself to a position before this message to process messages that could not be processed on a previous scan it will reprocess the failing message and will consequently reissue this message.

User action: Remove the invalid message from the dead letter queue. Do not write messages to the dead letter queue unless they have been prefixed by a valid MQDLH. If you require a dead letter queue handler that can process messages not prefixed by a valid MQDLH, you must change the sample program called amqsdlq to cater for your needs.

AMQ8722 Dead letter queue handler unable to put message: Rule &1 Reason &2.

Explanation: This message is produced by the dead letter queue handler when it is requested to redirect a message to another queue but is unable to do so. If the reason that the redirect fails is the same as the reason the message was put to the dead letter queue then it is assumed that no new error has occured and no message is produced. The retry count for the message will be incremented and the dead letter queue handler will continue.

User action: Investigate why the dead letter queue handler was unable to put the message to the dead letter queue. The line number of the rule used to determine the action for the message should be used to help identify to which queue the dead letter queue handler attempted to PUT the message.

AMQ8741 Unable to connect to queue manager(&3) : CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not connect to the requested queue manager. This message is typically issued when the requested queue manager has not been started or is quiescing, or if the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8742 Unable to open queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the queue manager object. This message is typically issued because of a resource shortage or because the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8743 Unable to inquire on queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not inquire on the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8744 Unable to close queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not close the queue manager. This

message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Take appropriate action based upon the completion code and reason.

AMQ8745 Unable to open dead letter queue(&3) for browse: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing. This message is typically issued because another process has opened the dead letter queue for exclusive access, or becuase an invalid dead letter queue name was specified. Other possible reasons include resource shortages or insufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Take appropriate action based upon the completion code and reason.

AMQ8746 Unable to close dead letter queue: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) could not close the dead letter queue. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Take appropriate action based upon the completion code and reason.

AMQ8747 Integer parameter(&2) outside permissable range for &3 on line &1.
Explanation: An integer supplied as input to the dead letter handler was outside of the valid range of values for a particular keyword.
User action: Correct the input data and restart the dead letter queue handler.

AMQ8748 Unable to get message from dead letter queue: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue. This message is typically issued because of the queue manager ending, a resource problem, or another process having deleted the dead letter queue. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Take appropriate action based upon the completion code and reason.

AMQ8749 Unable to commit/backout action on dead letter queue: CompCode = &1 Reason
Explanation: The dead letter queue handler (runmqdlq) was unable to commit or backout an update to the dead letter queue. This message is typically issued because of the queue manager ending, or because of a resource shortage. If the queue manager has ended, the update to the dead letter queue (and any associated updates) will be backed out when the queue manager restarts. If the problem was due to a resource problem then the updates will be backed out when the dead letter queue handler terminates. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Take appropriate action based upon the completion code and reason.

AMQ8750 No valid input provided to runmqdlq.
Explanation: Either no input was provided to runmqdlq, or the input to runmqdlq contained no valid message templates. If input was provided to runmqdlq but was found to be invalid, earlier messages will have been produced explaining the cause of the error. The dead letter queue handler will ends.
User action: Correct the input data and restart the dead letter queue handler.

AMQ8751 Unable to obtain private storage.
Explanation: The dead letter queue handler (runmqdlq) was unable to obtain private storage. This problem would typically arise as a result of some more global problem. For example if there is a persistent problem that is causing messages to be written to the DLQ and the same problem (for example queue full) is preventing the dead letter queue handler from taking the requested action with the message, it is necessary for the dead letter queue handler to maintain a large amount of state data to remember the retry counts associated with each message, or if the dead letter queue contains a large number of messages and the rules table has directed the dead letter queue handler to ignore the messages.
User action: Investigate if some more global problem exists, and if the dead letter queue contains a large number of messages. If the problem persists contact your support center.

AMQ8752 Parameter(&3) exceeds maximum length on line &1.
Explanation: A parameter supplied as input to the dead letter handler exceeded the maximum length for parameters of that type.
User action: Correct the input data and restart the dead letter queue handler.

AMQ8753 Duplicate parameter(&3) found on line &1.
Explanation: Two or more parameters of the same type were supplied on a single input line to the dead letter queue handler.
User action: Correct the input and restart the dead letter queue handler.

AMQ8756 Error detected releasing private storage.
Explanation: The dead letter queue handler (runmqdlq) was informed of an error while attempting to release an area of private storage. The dead letter queue handler ends.
User action: This message should be preceeded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ8757 Integer parameter(&3) outside permissable range on line &1.
Explanation: An integer supplied as input to the dead letter handler was outside of the valid range of integers supported by the dead letter queue handler.
User action: Correct the input data and restart the dead letter queue handler.

AMQ8758 &1 errors detected in input to runmqdlq.
Explanation: One or more errors have been detected in the input to the dead letter queue handler(runmqdlq). Error messages will have been generated for each of these errors. The dead letter queue handler ends.
User action: Correct the input data and restart the dead letter queue handler.

AMQ8759 Invalid combination of parameters to dead letter queue handler on line &1.
Explanation: An invalid combination of input parameters has been supplied to the dead letter queue handler. Possible causes are: no ACTION specified, ACTION(FWD) but no FWDQ specified, HEADER(YES|NO) specified without ACTION(FWD).
User action: Correct the input data and restart the dead letter queue handler.

AMQ8760 Unexpected failure while initializing process: Reason = &1.
Explanation: The dead letter queue handler (runmqdlq) could not perform basic initialization required to use MQ services because of an unforseen error. The dead letter queue handler ends.
User action: Use the standard facilities supplied with your system to record the problem identifier

and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8761 Unexpected failure while connecting to queue manager: CompCode = &1 Reason

Explanation: The dead letter queue handler (runmqdlq) could not connect to the requested queue manager because of an unforseen error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8762 Unexpected error while attempting to open queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the queue manager because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8763 Unexpected error while inquiring on queue manager: CompCode = &1 Reason = &

Explanation: The dead letter queue handler (runmqdlq) could not inquire on the queue manager because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8764 Unexpected error while attempting to close queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not close the queue manager because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8765 Unexpected failure while opening dead letter queue for browse: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8766 Unexpected error while closing dead letter queue: CompCode = &1 Reason = &2

Explanation: The dead letter queue handler (runmqdlq) could not close the dead letter queue because of an unforseen error. The completion code and the reason can be used to identify the

error. The dead letter queue handler ends.
User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8767 Unexpected error while getting message from dead letter queue: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8768 Unexpected error committing/backing out action on dead letter queue: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) was unable to either commit or backout an update to the dead letter queue because of an unforseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8769 Unable to disconnect from queue manager: CompCode = &1 Reason = &2.
Explanation: The dead letter queue handler (runmqdlq) was unable to disconnect from the queue manager because of an unexpected error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.
User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

Remote messages

AMQ9001 Channel program ended normally.
Explanation: Channel program '&3' ended normally.
User action: None.

AMQ9002 Channel program started.
Explanation: Channel program '&3' started.
User action: None.

AMQ9181 The response set by the exit is not valid.
Explanation: The user exit '&3' returned a response code '&1' that is not valid in the ExitResponse field of the channel exit parameters (MQCXP). Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set a response code that is not valid.

AMQ9182 The secondary response set by the exit is not valid.
Explanation: The user exit '&3' returned a secondary response code '&1' in the ExitResponse2 field of the channel exit parameters (MQCXP) that is not valid. Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set a secondary response code that is not valid.

AMQ9184 The exit buffer address set by the exit is not valid.
Explanation: The user exit '&3' returned an address '&1' for the exit buffer that is not valid, when the secondary response code in the ExitResponse2 field of the channel exit parameters (MQCXP) is set to MQXR2_USE_EXIT_BUFFER. Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set an exit buffer address that is not valid. The most likely cause is the failure to set a value, so that the value is 0.

AMQ9189 The data length set by the exit is not valid.
Explanation: The user exit '&3' returned a data length value '&1' that was not greater than zero. Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set a data length that is not valid.

AMQ9190 Channel stopping because of an error in the exit.
Explanation: The user exit '&3', invoked for channel '&4' with id '&1' and reason '&2', returned values that are not valid, as reported in the preceding messages. The channel stops.
User action: Investigate why the user exit program set values that are not valid.

AMQ9196 Data length is larger than the agent buffer length.
Explanation: The data length '&1' set by exit '&3' is larger than the agent buffer length. The user exit returned data in the supplied agent buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set a data length that is not valid..

AMQ9197 Data length is larger than the exit buffer length.
Explanation: The data length '&1' set by exit '&3' is larger than the exit buffer length. The user exit returned data in the supplied exit buffer, but the length specified is greater than the length of the

buffer. Message AMQ9190 is issued giving more details, and the channel stops.
User action: Investigate why the user exit program set a data length that is not valid.

AMQ9201 Allocate failed to host '&3'.
Explanation: The attempt to allocate a conversation using &4 to host '&3' was not successful.
User action: The error may be due to an incorrect entry in the &4 parameters contained in the channel definition to host '&3'. Correct the error and try again. If the error persists, record the error values and contact your systems administrator. The return code from &4 was &1 (X'&2'). It may be possible that the listening program at host '&3' is not running. If this is the case, perform the relevant operations to start the listening program for protocol &4 and try again.

AMQ9202 Remote host '&3' not available, retry later.
Explanation: The attempt to allocate a conversation using &4 to host '&3' was not successful. However the error may be a transitory one and it may be possible to successfully allocate a &4 conversation later.
User action: Try the connection again later. If the failure persists, record the error values and contact your systems administrator. The return code from &4 is &1 (X'&2'). The reason for the failure may be that this host cannot reach the destination host. It may also be possible that the listening program at host '&3' was not running. If this is the case, perform the relevant operations to start the &4 listening program, and try again.

AMQ9203 A configuration error for &4 occurred.
Explanation: Allocation of a &4 conversation to host '&3' was not possible.
User action: The configuration error may be one of the following: 1. If the communications protocol is LU6.2, it may be that one of the transmission parameters (Mode, or TP Name) is incorrect. Correct the error and try again. The mode name should be the same as the mode defined on host &3. The TP name on &3 should be defined. 2. If the communications protocol is LU6.2, it may be that an LU6.2 session has not been established. Contact your systems administrator. 3. If the communications protocol is TCP/IP, it may be that the host name specified is incorrect. Correct the error and try again. 4. If the communications protocol is TCP/IP, it may be that the host name specified cannot be resolved to a network address. The host name may not be in the nameserver. The return code from &4 is &1 (X'&2'). Record the error values and tell the system administrator.

AMQ9204 Connection to host '&3' rejected.
Explanation: Connection to host '&3' over &4 was rejected.
User action: The remote system might not be configured to allow connections from this host. Check the &4 listener program has been started on host '&3'. If the conversation uses LU6.2, it is possible that either the userid or password supplied to the remote host is incorrect. If the conversation uses TCP/IP, it is possible that the remote host does not recognize the local host as a valid host. The return code from &4 is &1 X('&2'). Record the values and tell the systems administrator.

AMQ9205 The host name supplied is not valid.
Explanation: The supplied &4 host name '&3' could not be resolved into a network address. Either the name server does not contain the host, or the name server was not available.
User action: Check the &4 configuration on your host.

AMQ9206 Error on send to host '&3'.
Explanation: An error occurred sending data over &4 to '&3'. This may be due to a

communications failure.
User action: Record the value &1 and the return code &4 and tell your systems administrator.

AMQ9207 The data received from host '&3' is not valid.

Explanation: Incorrect data format received from host '&3' over &4. It may be that an unknown host is attempting to send data. An FFST file has been generated containing the invalid data received.
User action: Tell the systems administrator.

AMQ9208 Error on receive from host '&3'.

Explanation: An error occurred receiving data from '&3' over &4. This may be due to a communications failure.
User action: Record the &4 return code &1 (X'&2') and tell the systems administrator.

AMQ9209 Connection to host '&3' closed.

Explanation: An error occurred receiving data from '&3' over &4. The connection to the remote host has unexpectedly terminated.
User action: Tell the systems administrator.

AMQ9210 Remote attach failed.

Explanation: There was an incoming attach from a remote host but the local host could not complete the bind.
User action: Record the &4 return code &1 (X'&2') and tell the systems administrator who should check the &4 configuration.

AMQ9211 Error allocating storage.

Explanation: The program was unable to obtain enough storage.
User action: Stop some programs which are using storage and retry the operation. If the problem persists contact your Systems Administrator.

AMQ9212 A TCP/IP socket could not be allocated.

Explanation: A TCP/IP socket could not be created, possibly because of a storage problem.
User action: Try the program again. If the failure persists record the value &1 and tell the systems administrator.

AMQ9213 A communications error for &4 occurred.

Explanation: An unexpected error occurred in communications.
User action: The return code from the &4&3 call was &1 (X'&2'). Record these values and tell the systems administrator.

AMQ9214 Attempt to use an unsupported communications protocol.

Explanation: An attempt was made to use an unsupported communications protocol type &2.
User action: Check the channel definition file. It may be that the communications protocol entered is not a currently supported one.

AMQ9215 Communications subsystem unavailable.

Explanation: An attempt was made to use the communications subsystem, but it has not been started.
User action: Start the communications subsystem, and rerun the program.

AMQ9216 Usage: &3 [-m QMgrName] [-n TPName]
Explanation: Values passed to the responder channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-n TPName] Default values will be used for parameters not supplied.
User action: Correct the parameters passed to the Channel program and retry the operation.

AMQ9217 The TCP/IP listener program could not be started.
Explanation: An attempt was made to start a new instance of the listener program, but the program was rejected.
User action: The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the listener program later).

AMQ9218 The TCP/IP listener program could not bind to port number &1.
Explanation: An attempt to bind the TCP/IP socket to the listener port was unsuccessful.
User action: The failure could be due to another program using the same port number. Record the return code &2 from the bind and tell the systems administrator.

AMQ9219 The TCP/IP listener program could not create a new connection for the incoming conversation.
Explanation: An attempt was made to create a new socket because an attach request was received, but an error occurred.
User action: The failure may be transitory, try again later. If the problem persists, record the return code &1 and tell the systems administrator. It may be necessary to free some jobs, or restart the communications system.

AMQ9220 The &4 communications program could not be loaded.
Explanation: The attempt to load the &4 library or procedure '&3' failed with error code &1.
User action: Either the library must be installed on the system or the environment changed to allow the program to locate it.

AMQ9221 Unrecognized protocol was specified.
Explanation: The specified value of '&3' was not recognized as one of the protocols supported.
User action: Correct the parameter and retry the operation.

AMQ9222 Cannot find the configuration file.
Explanation: The configuration file '&3' cannot be found. This file contains default definitions for communication parameters. Default values will be used.
User action: None.

AMQ9223 Enter a protocol type.
Explanation: The operation you are performing requires that you enter the type of protocol.
User action: Add the protocol parameter and retry the operation.

AMQ9224 Unexpected token detected.
Explanation: On line &1 of the INI file keyword '&3' was read when a keyword was expected.
User action: Correct the file and retry the operation.

AMQ9225 File syntax error.
Explanation: A syntax error was detected on line &1 while processing the INI file.

User action: Correct the problem and retry the operation.

AMQ9226 Usage: &3 [-m QMgrName] -t (TCP | LU62 | NETBIOS) [ProtocolOptions]

Explanation: Values passed to the listener program were invalid. The parameter string passed to this program is as follows :- [-m QMgrName] (-t TCP [-p Port] | -t LU62 [-n TPName] | -t NETBIOS [-l LocalName] [-e Names] [-s Sessions] [-o Commands] [-a Adaptor]) Default values will be used for parameters not supplied.

User action: Correct the parameters passed to the listener program and retry the operation.

AMQ9227 &3 local host name not provided.

Explanation:

User action: Add a local name to the configuration file and retry the operation.

AMQ9228 The &4 responder program could not be started.

Explanation: An attempt was made to start an instance of the responder program, but the program was rejected.

User action: The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the responder program later).

AMQ9229 The application has been ended.

Explanation: You have issued a request to end the application.

User action: None.

AMQ9230 An unexpected &4 event occurred.

Explanation: During the processing of network events, an unexpected event &1 occurred.

User action: None.

AMQ9231 The supplied parameter is not valid.

Explanation: The value of the &4 &5 parameter has the value '&3'. This value has either not been specified or has been specified incorrectly.

User action: Check value of the &5 parameter and correct it if necessary. If the fault persists, record the return code (&1,&2) and &4 and tell the systems administrator.

AMQ9232 No &3 specified

Explanation: The operation requires the specification of the &3 field.

User action: Specify the &3 and retry the operation.

AMQ9233 Error creating Listener thread for &3.

Explanation: The process attempted to create a new thread for an incoming connection.

User action: Contact the systems administrator.

AMQ9235 The supplied Local LU was invalid.

Explanation: The &4 Local LU name '&3' was invalid.

User action: Either the Local LU name was entered incorrectly or it was not in the &4 communications configuration. Correct the error and try again.

AMQ9236 The supplied Partner LU was invalid.

Explanation: The &4 Partner LU name '&3' was invalid.

User action: Either the Partner LU name was entered incorrectly or it was not in the &4

communications configuration. Correct the error and try again.

AMQ9501 Usage: &3 [-m QMgrName] -c ChlName.
Explanation: Values passed to the channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] -c ChlName Default values will be used for parameters not supplied.
User action: Correct the parameters passed to the Channel program and retry the operation.

AMQ9502 Type of channel not suitable for action requested.
Explanation: The operation requested cannot be performed on channel '&3'. Some operations are only valid for certain channel types. For example, you can only ping a channel from the end sending the message.
User action: Check whether the channel name is specified correctly. If it is check that the channel has been defined correctly.

AMQ9503 Channel negotiation failed.
Explanation: Channel '&3' between this machine and the remote machine could not be established due to a negotiation failure.
User action: Tell the systems administrator who should look at the log on the remote system for messages explaining the cause of the negotiation failure.

AMQ9504 A protocol error was detected for channel '&3'.
Explanation: During communications with the remote queue manager, the channel program detected a protocol error. The failure type was &1 with associated data of &2.
User action: Contact the systems administrator who should examine the error logs to determine the cause of the failure.

AMQ9505 Channel sequence number wrap values are different.
Explanation: The sequence number for channel '&3' is &1, but the value specified at the remote location is &2. The two values must be the same before the channel can be started.
User action: Change either the local or remote channel definitions so that the values specified for the message sequence number wrap values are the same.

AMQ9506 Message receipt confirmation failed.
Explanation: Channel '&3' has ended because the remote queue manager did not accept the last batch of messages.
User action: The error log for the channel at the remote site will contain an explanation of the failure. Contact the remote Systems Administrator to resolve the problem.

AMQ9507 Channel '&3' is currently in-doubt.
Explanation: The requested operation cannot complete because the channel is in-doubt with host '&4'.
User action: Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or use the RESOLVE CHANNEL command to correct the problem manually.

AMQ9508 Program cannot connect to the queue manager.
Explanation: The connection attempt to queue manager '&4' failed with reason code &1.
User action: Ensure that the queue manager is available and operational.

AMQ9509 Program cannot open queue manager object.

Explanation: The attempt to open either the queue or queue manager object '&4' on queue manager '&5' failed with reason code &1.
User action: Ensure that the queue is available and retry the operation.

AMQ9510 Messages cannot be retrieved from a queue.
Explanation: The attempt to get messages from queue '&4' on queue manager '&5' failed with reason code &1.
User action: Ensure that the required queue is available and operational.

AMQ9511 Messages cannot be put to a queue.
Explanation: The attempt to put messages to queue '&4' on queue manager '&5' failed with reason code &1.
User action: Ensure that the required queue is available and operational.

AMQ9512 Ping operation is not valid for channel '&3'.
Explanation: Ping may only be issued for SENDER or SERVER channel types.
User action: If the local channel is a receiver channel, you must issue the ping from the remote queue manager.

AMQ9513 Maximum number of channels reached.
Explanation: The maximum number of channels that can be in use simultaneously has been reached.
User action: Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The number of permitted channels is a configurable parameter in the queue manager configuration file.

AMQ9514 Channel '&3' is in use.
Explanation: The requested operation failed because channel '&3' is currently active.
User action: Either end the channel manually, or wait for it to close, and retry the operation.

AMQ9515 Channel '&3' changed.
Explanation: The statistics shown are for the channel requested, but it is a new instance of the channel. The previous channel instance has ended.
User action: None.

AMQ9516 File error occurred.
Explanation: The filesystem returned error code &1 for file '&3'.
User action: Record the name of the file '&3' and tell the systems administrator, who should ensure that file '&3' is correct and available.

AMQ9517 File damaged.
Explanation: The program has detected damage to the contents of file '&3'.
User action: Record the values and tell the systems administrator who must restore a saved version of file '&3'. The return code was '&1' and the record length returned was '&2'.

AMQ9518 File '&3' not found.
Explanation: The program requires that the file '&3' is present and available.
User action: Record the name of the file and tell the systems administrator who must ensure that file '&3' is available to the program.

AMQ9519 Channel '&3' not found.
Explanation: The requested operation failed because the program could not find a definition of channel '&3'.
User action: Check that the name is specified correctly and the channel definition is available.

AMQ9520 Channel not defined remotely.
Explanation: There is no definition of channel '&3' at the remote location.
User action: Add an appropriate definition to the remote hosts list of defined channels and retry the operation.

AMQ9521 Host is not supported by this channel.
Explanation: The connection across channel '&5' was refused because the remote host '&4' did not match the host '&3' specified in the channel definition.
User action: Update the channel definition, or remove the explicit mention of the remote machine connection name.

AMQ9522 Error accessing the status table.
Explanation: The program could not access the channel status table.
User action:

AMQ9523 Remote host detected a protocol error.
Explanation: During communications through channel '&3', the remote queue manager channel program detected a protocol error. The failure type was &1 with associated data of &2.
User action: Tell the systems administrator, who should examine the error files to determine the cause of the failure.

AMQ9524 Remote queue manager unavailable.
Explanation: Channel '&3' cannot start because the remote queue manager is not currently available.
User action: Either start the remote queue manager, or retry the operation later.

AMQ9525 Remote queue manager is ending.
Explanation: Channel '&3' is closing because the remote queue manager is ending.
User action: None.

AMQ9526 Message sequence number error for channel '&3'.
Explanation: The local and remote queue managers do not agree on the next message sequence number. A message with sequence number &1 has been sent when sequence number &2 was expected.
User action: Determine the cause of the inconsistency. It could be that the synchronization information has become damaged, or has been backed out to a previous version. If the situation cannot be resolved, the sequence number can be manually reset at the sending end of the channel using the RESET CHANNEL command.

AMQ9527 Cannot send message through channel '&3'.
Explanation: The channel has closed because the remote queue manager cannot receive a message.
User action: Contact the systems administrator who should examine the error files of the remote queue manager, to determine why the message cannot be received, and then restart the channel.

AMQ9528 User requested closure of channel '&3'.

Explanation: The channel is closing because of a request by the user.
User action: None.

AMQ9529 Target queue unknown on remote host.
Explanation: Communication using channel '&3' has ended because the target queue for a message is unknown at the remote host.
User action: Ensure that the remote host contains a correctly defined target queue, and restart the channel.

AMQ9530 Program could not inquire queue attributes.
Explanation: The attempt to inquire the attributes of queue '&4' on queue manager '&5' failed with reason code &1.
User action: Ensure that the queue is available and retry the operation.

AMQ9531 Transmission queue specification error.
Explanation: Queue '&4' identified as a transmission queue in the channel definition '&3' is not a transmission queue.
User action: Ensure that the queue name is specified correctly. If so, alter the queue usage parameter of the queue to that of a transmission queue.

AMQ9532 Program cannot set queue attributes.
Explanation: The attempt to set the attributes of queue '&4' on queue manager '&5' failed with reason code &1.
User action: Ensure that the queue is available and retry the operation.

AMQ9533 Channel '&3' is not currently active.
Explanation: The channel was not stopped because it was not currently active.
User action: None.

AMQ9534 Channel '&3' is currently not enabled.
Explanation: The channel program ended because the channel is currently not enabled.
User action: Issue the START CHANNEL command to re-enable the channel.

AMQ9535 User exit not valid.
Explanation: Channel program '&3' ended because user exit '&4' is not valid.
User action: Ensure that the user exit is specified correctly in the channel definition, and that the user exit program is correct and available.

AMQ9536 Channel ended by an exit.
Explanation: Channel program '&3' was ended by exit '&4'.
User action: None.

AMQ9537 Usage: &3 [-m QMgrName] [-q InitQ]
Explanation: Values passed to the Channel initiator program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-q InitQ] Default values will be used for parameters not supplied.
User action: Correct the parameters passed to the program and retry the operation.

AMQ9538 Commit control error.
Explanation: An error occurred when attempting to start commitment control. Either exception

'&3' was received when querying commitment status, or commitment control could not be started.
User action: Refer to the error log for other messages pertaining to this problem.

AMQ9539 No channels available.
Explanation: The channel initiator program received a trigger message to start an MCA program to process queue '&3'. The program could not find a defined, available channel to start.
User action: Ensure that there is a defined channel, which is enabled, to process the transmission queue.

AMQ9540 Commit failed.
Explanation: The program ended because return code &1 was received when an attempt was made to commit change to the resource managers. The commit ID was '&3'.
User action: Tell the systems administrator.

AMQ9541 CCSID supplied for data conversion not supported.
Explanation: The program ended because, either the source CCSID '&1' or the target CCSID '&2' is not valid, or is not currently supported.
User action: Correct the CCSID that is not valid, or ensure that the requested CCSID can be supported.

AMQ9542 Queue manager is ending.
Explanation: The program will end because the queue manager is quiescing.
User action: None.

AMQ9543 Status table damaged.
Explanation: The channel status table has been damaged.
User action: End all running channels and issue a DISPLAY CHSTATUS command to see the status of the channels. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your Willow Technology support center. Do not discard these files until the problem has been resolved.

AMQ9544 Messages written to the 'dead-letter queue'.
Explanation: During the processing of channel '&3' one or more messages have been put to a dead-letter queue. The location of the messages is &1, where 1 is the local dead-letter queue and 2 is the remote dead-letter queue.
User action: Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed. The program identifier (PID) of the processing program was '&4'.

AMQ9545 Disconnect interval expired.
Explanation: Channel '&3' closed because no messages arrived on the transmission queue within the disconnect interval period.
User action: None.

AMQ9546 Error return code received.
Explanation: The program has ended because return code &1 was returned from an internal function.
User action: Correct the reason for the failure and retry the operation.

AMQ9547 Type of remote channel not suitable for action requested.

Explanation: The operation requested cannot be performed because channel '&3' on the remote machine is not of a suitable type. For example, if the local channel is defined as a sender the remote machine must define its channel as either a receiver or requester.
User action: Check that the channel name is specified correctly. If it is, check that the remote channel has been defined correctly.

AMQ9548 Message put to the 'dead-letter queue'.
Explanation: During processing a message has been put to the dead-letter queue.
User action: Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

AMQ9549 Transmission Queue '&3' inhibited for MQGET.
Explanation: An MQGET failed because the transmission queue had been previously inhibited for MQGET.
User action: None.

AMQ9550 Channel program &3 cannot be stopped at this time.
Explanation: The channel program is currently busy and cannot be stopped at the moment.
User action: Issue the STOP CHANNEL command again at a later time.

AMQ9551 Protocol not supported by remote host
Explanation: The operation you are performing over Channel '&3' to the host at '&4' is not supported by the target host.
User action: Check that the connection name parameter is specified correctly and that the levels of the products in use are compatible.

AMQ9552 Security flow not received.
Explanation: During communications through channel '&3' the local security exit requested security data from the remote machine. The security data has not been received so the channel has been closed.
User action: Tell the systems administrator who should ensure that the security exit on the remote machine is defined correctly.

AMQ9553 Not supported.
Explanation: The command or function attempted is not currently supported on this platform.
User action: None.

AMQ9554 User not authorized.
Explanation: You are not authorized to perform the Channel operation.
User action: Tell the systems administrator who should ensure that the correct access permissions are available to you, and then retry the operation.

AMQ9555 File format error.
Explanation: The file '&3' does not have the expected format.
User action: Ensure that the file name is specified correctly.

AMQ9556 Channel synchronization file missing or damaged.
Explanation: The channel synchronization file '&3' is missing or does not correspond to the stored channel information for queue manager '&4'.

User action: Rebuild the synchronization file using the rcrmqobj command rcrmqobj -t syncfile (-m q-mgr-name)

AMQ9557 Queue Manager UserID initialization failed.
Explanation: The call to initialize the user ID failed with CompCode &1 and Reason &2.
User action: Correct the error and try again.

AMQ9558 Remote Channel is not currently available.
Explanation: The channel program ended because the channel '&3' is not currently available on the remote system. This could be because the channel is disabled or that the remote system does not have sufficient resources to run a further channel.
User action: Check the remote system to ensure that the channel is available to run and retry the operation.

AMQ9560 Rebuild Synchronization File - program started
Explanation: Rebuilding the Synchronization file for Queue Manager '&3' .
User action: None.

AMQ9561 Rebuild Synchronization File - program completed normally
Explanation: Rebuild Synchronization File program completed normally.
User action: None.

AMQ9562 Synchronization file in use.
Explanation: The Synchronization file '&3' is in use and cannot be recreated.
User action: Stop any channel activity and retry the rcrmqobj command.

AMQ9563 Synchronization file cannot be deleted
Explanation: The filesystem returned error code &1 for file '&3'.
User action: Tell the systems administrator who should ensure that file '&3' is available and not in use.

AMQ9564 Synchronization File cannot be created
Explanation: The filesystem returned error code &1 for file '&3'.
User action: Tell the systems administrator.

AMQ9565 No dead-letter queue defined.
Explanation: The queue manager '&4' does not have a defined dead-letter queue.
User action: Either correct the problem that caused the program to try and write a message to the dead-letter queue or create a dead-letter queue for the queue manager.

AMQ9566 Invalid MQSERVER value
Explanation: The value of the MQSERVER environment variable was '&3'. The variable should be in the format 'ChannelName/Protocol/ConnectionName'.
User action: Correct the MQSERVER value and retry the operation.

AMQ9572 Message header is not valid.
Explanation: Channel '&3' is stopping because a message header is not valid. During the processing of the channel, a message was found that has a header that is not valid. The dead-letter queue has been defined as a transmission queue, so a loop would be created if the message had been put there.
User action: Correct the problem that caused the message to have a header that is not valid.

AMQ9573 Maximum number of active channels reached.
Explanation: There are too many channels active to start another. The current defined maximum number of active channels is &1.
User action: Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The maximum number of active channels is a configurable parameter in the queue manager configuration file.

AMQ9574 Channel &3 can now be started.
Explanation: Channel &3 has been waiting to start, but there were no channels available because the maximum number of active channels was running. One, or more, of the active channels has now closed so this channel can start.
User action:

AMQ9999 Channel program ended abnormally.
Explanation: Channel program '&3' ended abnormally.
User action: Look at previous error messages for channel program '&3' in the error files to determine the cause of the failure.

Chapter 16. 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

administrator commands

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

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.

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 open an MQSeries object.

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, and MQSeries for Windows NT, 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

back out

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.

basic mapping support (BMS)

An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS

Basic mapping support.

browse

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

browse cursor

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

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

checkpoint

(1) A time when significant information is written on the log. Contrast with *syncpoint.*

(2) 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.

checkpoint

A time when significant information is written on the log. Contrast with *syncpoint.*

circular logging

In MQSeries on UNIX systems, MQSeries for OS/2, 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.*

coded character set identifier (CCSID)

The name of a coded set of characters and their code point assignments.

command

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

command prefix (CPF)

In MQSeries for MVS/ESA, a character string that identifies the queue manager to which MQSeries for MVS/ESA commands are directed, and from which MQSeries for MVS/ESA operator messages are received.

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.

completion code

A return code indicating how an MQI call has ended.

configuration file

In MQSeries on UNIX systems, MQSeries for OS/2, 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, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. These commands require only that the MQSeries product be installed; they do not require a special utility or program to run them.

controlled shutdown

See *quiesced shutdown.*

D

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

In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ

Dead-letter queue.

dynamic queue

A local queue that is 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 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.

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, MQSeries for Windows NT, and MQSeries for OS/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

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*.

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, 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.

L

linear logging

In MQSeries on UNIX systems, MQSeries for OS/2, 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 information about incoming network connections.

local definition

An MQSeries object that belongs to a local queue manager.

local definition of a remote queue

An MQSeries object that belongs 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, records the work done by queue managers while they receive, transmit, and deliver messages.

log control file

In MQSeries on UNIX systems, MQSeries for OS/2, 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, 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*.

M

MCA

Message channel agent.

MCI

Message channel interface.

media image

In MQSeries on UNIX systems, MQSeries for OS/2, 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

(1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*.

(2) In system programming, information intended for the terminal operator or system administrator.

message channel

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

message channel agent (MCA)

A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

message 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.

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 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

name service

In MQSeries for AIX, MQSeries for OS/2, 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, 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.

nonpersistent message

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

NSI

Name service interface.

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 (MVS/ESA only), or a storage class (MVS/ESA only).

Object authority manager (OAM)

In MQSeries on UNIX systems 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 (MQOD). 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.

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 that indicates 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 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, and MQSeries for Windows NT, a term used for a user identifier (ID). 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 that is used by:

  • User administration applications that 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 Willow Technology 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

(1) 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*.

(2) 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, an error condition caused by a queue being 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

(1) In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*.

(2) 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.

remote queue

A queue that belongs 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 is remote if it is not the queue manager to which the program is connected.

remote queue object

See *local definition of a remote queue.*

remote queuing

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

reply message

A type of message used for replies to request messages.

reply-to queue

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

report message

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

requester channel

In message queuing, a 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 for requesting a reply from another program.

resolution path

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

resource manager

An application, program, or transaction that manages 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.

shutdown

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

single-phase backout

A method in which an action that is 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.*

SIT

System initialization table.

stanza

A group of lines in a configuration file that assigns a value to a parameter that modifies the behavior of a queue manager, client, or channel. In MQSeries on UNIX systems, MQSeries for OS/2, 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.

synchronous messaging

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

syncpoint

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

system.command.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.

system initialization table (SIT)

A table containing parameters used by CICS on start up.

T

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 identifier

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

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 that allows a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message

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

trigger monitor

A continuously-running application that serves 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

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.*

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.