MQSeries®

System Administration

SC33-1873-01

MQSeries®

System Administration

SC33-1873-01

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix I, “Notices” on page 379.

Second edition (March 1999)

This edition applies to the following products:

- MQSeries for AIX® V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2® Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT® V5.1

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

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

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

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

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

Contents

Figures

Tables

Tables

About this book

This book applies to the MQSeries Version 5 products, which are:

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

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 the MQSeries Version 5 products, and the services they provide to support commercial messaging. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Installation of MQSeries is described in one of the following:

- “Chapter 3. Installing the MQSeries for AIX Server” in the *MQSeries for AIX V5.1 Quick Beginnings* book
- “Chapter 3. Installing MQSeries for OS/2 Warp” in the *MQSeries for OS/2 Warp V5.1 Quick Beginnings* book
- “Chapter 3. Installing the MQSeries for HP-UX Server” in the *MQSeries for HP-UX V5.1 Quick Beginnings* book
- “Chapter 3. Installing the MQSeries for Sun Solaris Server” in the *MQSeries for Sun Solaris V5.1 Quick Beginnings* book
- Chapter 4, “Installing MQSeries for Windows NT” in the *MQSeries for Windows NT V5.1 Quick Beginnings* book

Post-installation configuration of a distributed queuing network is described in Chapter 2, “Making your applications communicate” in the *MQSeries Intercommunication* book.

Who this book is for

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

What you need to know to understand this book

To use this book, you should have a good understanding of the operating systems described here, and of the utilities associated with them. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

Terms used in this book

In this book, the term "the **MQSeries Version 5 products**" means:

| MQSeries for AIX V5.1
| MQSeries for HP-UX V5.1
| MQSeries for OS/2 Warp V5.1
| MQSeries for Sun Solaris V5.1
| MQSeries for Windows NT V5.1

The term "**MQSeries for UNIX® systems**" means:

| MQSeries for AIX V5.1
| MQSeries for HP-UX V5.1
| MQSeries for Sun Solaris V5.1

Using MQSeries for UNIX systems

The following restrictions apply to the use of UNIX operating-system facilities with the MQSeries product:

1. MQSeries for AIX and MQSeries for HP-UX use the UNIX subroutine `ftok` to generate standard interprocess communication keys. Using `ftok` exclusively within a node ensures that these keys are unique, which is a requirement of MQSeries. Therefore, do not use any code that generates interprocess keys in a different way.
2. MQCONN sets up its own signal handler for the signals:

   SIGSEGV
   SIGBUS

   User handlers for these signals are restored after every MQI call.

   The remaining signals are handled differently.

   SIGINT
   SIGQUIT
   SIGFPE
   SIGTERM
   SIGHUP

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

3. 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.

The base directory

Throughout this book, the name **mqmtop** has been used to represent the name of the base directory where MQSeries is installed on UNIX systems.

- For MQSeries for AIX, **mqmtop** represents the directory **/usr/mqm**.
- For other UNIX systems, the name of the actual directory is **/opt/mqm**.

Using MQSeries for OS/2 Warp and Windows NT

Examples in this book relevant to MQSeries for Windows NT may use New Technology file system (NTFS), high performance file system (HPFS), or file allocation table (FAT) file names. Examples relevant to MQSeries for OS/2 Warp may use HPFS or FAT file names.

The examples are valid for all file-naming systems, the name being transformed if necessary when the FAT system is in use. Name transformation is described in “Understanding MQSeries file names” on page 27.

The calls MQCONN and MQCONNX

References in this book to the call MQCONN - Connect queue manager can be replaced by references to the call MQCONNX - Connect queue manager (extended); MQCONNX requires an additional parameter. For more information about these calls, see “MQCONN - Connect queue manager” and “MQCONNX - Connect queue manager (extended)” in the *MQSeries Application Programming Reference* manual.

MQSeries publications

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

MQSeries cross-platform publications

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

- MQSeries for AIX V5.1
- MQSeries for AS/400® V4R2M1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Digital OpenVMS V2.2
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390® V2.1
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Sun Solaris V5.1
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries for VSE/ESA™ V2.1
- MQSeries for Windows™ V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.1

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in "MQSeries Level 1 product publications" on page xx. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

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

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

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

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

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

MQSeries System Administration
The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem determination, and the dead-letter queue handler. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

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

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

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

MQSeries Messages
The *MQSeries Messages* book, GC33-1876, which describes “AMQ” messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

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

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

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

MQSeries Using C++
MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V4R2M1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

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

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

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

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

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

This book applies to the following MQSeries products only:

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

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

This book applies to the following MQSeries products only:

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

MQSeries platform-specific publications

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

MQSeries for AIX

MQSeries for AIX Version 5 Release 1 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 Version 4 Release 2.1 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX

MQSeries for AT&T GIS UNIX Version 2 Release 2 System Management Guide, SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS Version 2 Release 2 System Management Guide, GC33-1791

MQSeries for Digital UNIX

MQSeries for Digital UNIX Version 2 Release 2.1 System Management Guide, GC34-5483

MQSeries for HP-UX

MQSeries for HP-UX Version 5 Release 1 Quick Beginnings, GC33-1869

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp Version 5 Release 1 Quick Beginnings, GC33-1868

MQSeries for OS/390

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

MQSeries for OS/390 Version 2 Release 1 Program Directory

MQSeries for OS/390 Version 2 Release 1 System Management Guide, SC34-5374

MQSeries for OS/390 Version 2 Release 1 Messages and Codes, GC34-5375

MQSeries for OS/390 Version 2 Release 1 Problem Determination Guide, GC34-5376

MQSeries link for R/3

MQSeries link for R/3 Version 1 Release 2 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx Version 2 Release 2 System Management Guide, GC33-1768

MQSeries for Sun Solaris

MQSeries for Sun Solaris Version 5 Release 1 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2 Release 2 System Management Guide, GC33-1893

MQSeries for VSE/ESA

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

MQSeries for VSE/ESA Version 2 Release 1 System Management Guide, GC34-5364

MQSeries for Windows

MQSeries for Windows Version 2 Release 0 User's Guide, GC33-1822
MQSeries for Windows Version 2 Release 1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT Version 5 Release 1 Quick Beginnings, GC34-5389
MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387
MQSeries LotusScript® Extension, SC34-5404

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC33-1141

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for UnixWare Version 1 Release 4.1 User's Guide, SC33-1379

Softcopy books

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

BookManager format®

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

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

HTML format

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

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

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

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

Portable Document Format (PDF)

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

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

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

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

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

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

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

MQSeries Web site

The MQSeries product family Web site is at:

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

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Related publications

This section lists other documentation referred to in this book.

IBM TXSeries™ Administration Reference, SC33-1563

IBM TXSeries for Windows NT, Version 4.2: CICS® Administration Guide, SC33-1881

Summary of Changes

This edition of *MQSeries System Administration* applies to these new versions and releases of MQSeries products:

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

Major new function supplied with each of these MQSeries products is summarized here.

MQSeries V5.1

The MQSeries Version 5 Release 1 products are:

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

The following new function is provided in all of the V5.1 products:

MQSeries queue manager clusters

MQSeries queue managers can be connected to form a *cluster* of queue managers. Within a cluster, queue managers can make the queues they host available to every other queue manager. Any queue manager can send a message to any other queue manager in the same cluster without the need for explicit channel definitions, remote queue definitions, or transmission queues for each destination. The main benefits of MQSeries clusters are:

- Fewer system administration tasks
- Increased availability
- Workload balancing

Clusters are supported by these MQSeries products:

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

See the book *MQSeries Queue Manager Clusters*, SC34-5349, for a complete description of this function.

MQSeries Administration Interface (MQAI)

The MQSeries Administration Interface is an MQSeries programming interface that simplifies manipulation of MQSeries PCF messages for administrative tasks. It is described in a new book, *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390.

Support for Windows 98 clients
A Windows 98 client can connect to any MQSeries V5.1 server.

Message queue size
A message queue can be up to 2 GB.

Controlled, synchronous shutdown of a queue manager
A new option has been added to the **endmqm** command to allow controlled, synchronous shutdown of a queue manager.

Java support
The MQSeries Client for Java and MQSeries Bindings for Java are provided with all MQSeries V5.1 products. The client, bindings, and common files have been packaged into .jar files for ease of installation.

Euro support
MQSeries supports new and changed code pages that use the euro currency symbol. Details of code pages that include the euro symbol are provided in the *MQSeries Application Programming Reference* book.

Conversion of the EBCDIC new-line character
You can control the conversion of EBCDIC new-line characters to ensure that data transmitted from EBCDIC systems to ASCII systems and back to EBCDIC is unaltered by the ASCII conversion.

Client connections via MQCONNX
A client application can specify the definition of the client-connection channel at run time in the MQCNO structure of the MQCONNX call.

Additional new function in MQSeries for AIX V5.1

- The UDP transport protocol is supported.
- Sybase databases can participate in global units of work.
- Multithreaded channels are supported.

Additional new function in MQSeries for HP-UX V5.1

- MQSeries for HP-UX V5.1 runs on both HP-UX V10.20 and HP-UX V11.0.
- Multithreaded channels are supported.
- Both HP-UX kernel threads and DCE threads are supported.

Additional new function in MQSeries for OS/2 Warp V5.1

OS/2 high memory support is provided.

Additional new function in MQSeries for Sun Solaris V5.1

- MQSeries for Sun Solaris V5.1 runs on both Sun Solaris V2.6 and Sun Solaris 7.
- Sybase databases can participate in global units of work.
- Multithreaded channels are supported.

Additional new function in MQSeries for Windows NT V5.1

MQSeries for Windows NT V5.1 is part of the IBM Enterprise Suite for Windows NT. New function in this release includes:

- Close integration with Microsoft Windows NT Version 4.0, including exploitation of extra function provided by additional Microsoft offerings. The main highlights are:
  - Graphical tools and applications for managing, controlling, and exploring MQSeries:
    - MQSeries Explorer—a snap-in for the Microsoft management console (MMC) that allows you to query, change, and create the local, remote, and cluster objects across an MQSeries network.
    - MQSeries Services—an MMC snap-in that controls the operation of MQSeries components, either locally or remotely within the Windows NT domain. It monitors the operation of MQSeries servers and provides extensive error detection and recovery functions.
    - MQSeries API Exerciser—a graphical application for exploring the messaging and queuing programming functions that MQSeries provides. It can also be used in conjunction with the MQSeries Explorer to gain a deeper understanding of the effects of MQSeries operations on objects and messages.
    - MQSeries Postcard—a sample application that can be used to verify an MQSeries installation, for either local or remote messaging.
  - Support for the following features of Windows NT has been added:
    - Windows NT performance monitor—used to access and display MQSeries information, such as the current depth of a queue and the rate at which message data is put onto and taken off queues.
    - ActiveDirectory—programmable access to MQSeries objects is available through the Active Directory Service Interfaces (ADSI).
    - Windows NT user IDs—previous MQSeries restrictions on the validity of Windows NT user IDs have been removed. All valid Windows NT user IDs are now valid identifiers for MQSeries operations. MQSeries uses the associated Windows NT Security Identifier (SID) and the Security Account Manager (SAM). The SID allows the MQSeries Object Authority Manager (OAM) to identify the specific user for an authorization request.
    - Windows NT registry—now used to hold all configuration and related data. The contents of any configuration (.INI) files from previous MQSeries installations of MQSeries for Windows NT products are migrated into the registry; the .INI files are then deleted.
    - A set of Component Object Model (COM) classes, which allow ActiveX applications to access the MQSeries Message Queue Interface (MQI) and the MQSeries Administration Interface (MQAI).
  - An online Quick Tour of the product concepts and functions.

  - An online Information Center that gives you quick access to task help information, reference information, and Web-based online books and home pages.
  - Simplified installation of MQSeries for Windows NT, with default options and automatic configuration.
- Support for web-based administration of an MQSeries network, which provides a simplified way of using the MQSC commands and scripts and allows you to create powerful macros for standard administration tasks.
- Support for MQSeries LotusScript Extension (MQLSX), which allows Lotus Notes applications that are written in LotusScript to communicate with applications that run in non-Notes environments.
- Support for Microsoft Visual Basic for Windows Version 5.0.
- Performance improvements over the MQSeries for Windows NT Version 5.0 product.
- Information and examples on how MQSeries applications can interface with and exploit the lightweight directory access protocol (LDAP) directories.
- Support for Sybase participation in global units of work.

Part 1. Guidance

Chapter 1. Introduction to MQSeries

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

MQSeries and message queuing

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

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

The MQI is described in detail in Chapter 6, "Introducing the Message Queue Interface" in the *MQSeries Application Programming Reference* manual.

Time-independent applications

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

Message-driven processing

Upon arrival on a queue, messages can automatically start an application using a mechanism known as **triggering**. If necessary, the applications can be stopped when the message (or messages) have been processed.

Messages and queues

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

What is a message?

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

MQSeries messages have two parts:

- **The application data**
  The content and structure of the application data is defined by the application programs that use them.
- **A message descriptor**
  The message descriptor identifies the message and contains additional control information such as the type of message, and the priority assigned to the message by the sending application.

  The format of the message descriptor is defined by MQSeries. For a complete description of the message descriptor, see "MQMD - Message descriptor" in the *MQSeries Application Programming Reference* manual.

Message lengths

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

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

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

What is a queue?

A **queue** is a data structure used to store messages. The messages may be put on the queue by application programs, or by a **queue manager** as part of its normal operation.

Each queue is owned by a queue manager. The queue manager is responsible for maintaining the queues it owns and for storing all the messages it receives onto the appropriate queues.

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

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

How do applications send and receive messages?

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

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

1. Opens the required queue by issuing an MQI MQOPEN call
2. Issues an MQI MQPUT call to put the message onto the queue
3. Another application can retrieve the message from the same queue by issuing an MQI MQGET call.

For more information about MQI calls, see Chapter 3, “Call descriptions” in the *MQSeries Application Programming Reference* manual.

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

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

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

Retrieving messages from queues

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

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

The MQGET request from the application determines the method used.

Objects

Many of the tasks described in this book involve manipulating MQSeries **objects**.

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

The manipulation or *administration* of objects includes:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This is described in detail in "How to send a message to another queue manager" in the *MQSeries Intercommunication* book.
- Creating *clusters* of queue managers to simplify the overall administration process, or to achieve workload balancing.

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

Object names

The naming convention adopted for MQSeries objects depends on the object.

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

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

In MQSeries, names can have a maximum of 48 characters, with the exception of *channels* which have a maximum of 20 characters. For more information about names, see "Names of MQSeries objects" on page 279.

Managing objects

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

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

For more information about these methods, see Chapter 2, “An introduction to MQSeries administration” on page 23.

Object attributes

The properties of an object are defined by its attributes. Some you can specify, others you can only view. For example, the maximum message length that a queue can accommodate is defined by its *MaxMsgLength* attribute; you can specify this attribute when you create a queue. The *DefinitionType* attribute specifies how the queue was created; you can only display this attribute.

In MQSeries, there are 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 a queue that belongs to another queue manager.

A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager may exist on a remote machine across the network, or may exist on the same machine as the local queue manager.

MQSeries supports multiple queue managers on the same machine.

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

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

MQSeries queues

Queues are defined to MQSeries using:

- The appropriate MQSC DEFINE command
- The PCF Create Queue command

The commands specify the type of queue and its attributes. For example, a local queue object has attributes that specify what happens when applications reference that queue in MQI calls. Examples of attributes are:

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

For further details about defining queue objects, see “DEFINE QALIAS” through “DEFINE QREMOTE” in the *MQSeries Command Reference* manual or “Create Queue” in the *MQSeries Programmable System Management* manual.

Using queue objects

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

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

2. **A remote queue object**
   A remote queue object identifies a queue belonging to another queue manager. This queue must be defined as a local queue to that queue manager. The information you specify when you define a remote queue object allows the local queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.

   Before applications can send messages to a queue on another queue manager, you must have defined a transmission queue and channels between the queue managers, **unless** you have grouped one or more queue managers together into a *cluster*. For more information about clusters, see "Remote administration using clusters" on page 88.

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

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

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

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

Specific local queues used by MQSeries

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

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

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

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

If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager.

See “Managing objects for triggering” on page 80 and “runmqtrm (Start trigger monitor)” on page 326. For more information about triggering, see Chapter 14, “Starting MQSeries applications using triggers” in the *MQSeries Application Programming Guide*.

- **Transmission queues**
  Transmission queues are queues that temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration; see “Remote administration from a local queue manager using MQSC commands” on page 89. For information about the use of transmission queues in distributed queuing, see Chapter 1, “Concepts of intercommunication” in the *MQSeries Intercommunication* book.

- **Cluster transmission queues**
  Each queue manager within a cluster has a cluster transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. A definition of this queue is created by default on every queue manager on Version 5.1 of MQSeries for AIX, HP-UX, OS/2, Warp, Sun Solaris, and Windows NT.

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

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

  During name resolution, the cluster transmission queue takes precedence over the default transmission queue. When a queue manager that is not part of the cluster puts a message onto a remote queue, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use the default transmission queue.

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

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

  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 commands for processing. These commands are then retrieved by an MQSeries component called the command server. The command server validates the commands, passes the valid ones on for processing by the queue manager, and returns any responses to the appropriate reply-to queue.

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

- **Reply-to queues**
  When an application sends a request message, the application that receives the message can send back a reply message to the sending application. This message is put on a queue, called a reply-to queue, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.
- **Event queues**
  The MQSeries Version 5 products support instrumentation events, which can be used to monitor queue managers independently of MQI applications. Instrumentation events can be generated in several ways, for example:
  - An application attempting to put a message on a queue that is not available or does not exist.
  - A queue becoming full.
  - A channel being started.

  When an instrumentation event occurs, the queue manager puts an event message on an event queue. This message can then be read by a monitoring application which may inform an administrator or initiate some remedial action if the event indicates a problem. Note: Trigger events are quite different from instrumentation events in that trigger events are not caused by the same conditions, and do not generate event messages.

  For more information about instrumentation events, see Chapter 1, “Using instrumentation events to monitor queue managers” in the *MQSeries Programmable System Management* manual.

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries queue manager. See the “Initiation queues” entry under “Specific local queues used by MQSeries” on page 15 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 “Preparing channels and transmission queues for remote administration” on page 90 and “Message channels” in the *MQSeries Intercommunication* book,

Clusters

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

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

For information about clusters, see Chapter 9, “Administering remote MQSeries objects” on page 87 and the *MQSeries Queue Manager Clusters* book.

Namelists

A namelist is an MQSeries object that contains a list of other MQSeries objects. Typically, namelists are used by applications such as trigger monitors, where they are used to identify a group of queues. The advantage of using a namelist is that it is maintained independently of applications; that is, it can be updated without stopping any of the applications that use it. Also, if one application fails, the namelist is not affected and other applications can continue using it.

Namelists are also used with queue manager clusters so that you can maintain a list of clusters referenced by more than one MQSeries object.

System default objects

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

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

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

See Appendix A, “System and default objects” on page 343 for more information about system defaults.

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 supports client-server configurations for MQSeries applications.

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

An *MQI server* is a queue manager that provides queuing services to one or more clients. All the MQSeries objects, for example queues, exist only on the queue manager machine, that is, on the MQI server machine. A server can support normal local MQSeries applications as well.

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 Chapter 1, “Concepts of intercommunication” in the *MQSeries Intercommunication* book.

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

MQSeries applications in a client-server environment

When linked to a server, client MQSeries applications can issue most MQI calls in the same way as local applications. The client application issues an MQCONN call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager.

You must link your applications to the appropriate client libraries. See Chapter 10, “Using the message queue interface (MQI)” through Chapter 13, “Solving problems” in the *MQSeries Clients* book for further information.

Extending queue manager facilities

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

- User exits
- Installable services

User exits

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

- **Channel exits**
  These exits change the way that channels operate. Channel exits are described in Chapter 35, “Channel-exit programs” in the *MQSeries Intercommunication* book.
- **Data conversion exits**
  These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in Chapter 11, “Writing data-conversion exits” in the *MQSeries Application Programming Guide*.
- **The cluster workload exit**
  The function performed by this exit is defined by the provider of the exit. Call definition information is given in “MQWXP - Cluster workload exit parameter structure” in the *MQSeries Queue Manager Clusters* book. The exit is supported in the following environments: AIX, HP-UX, OS/2, Sun Solaris, Windows NT, and OS/390.

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 MQSeries product, or you can write your own component to perform the functions that you require.

Currently, the following installable services are provided:

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

The default service component that implements the service is the Object Authority Manager (OAM), which is supplied with MQSeries for UNIX systems and the MQSeries for Windows NT product. (The OAM is not supplied with MQSeries for OS/2 Warp.) By default, the OAM is active, and you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM. For more information about the OAM, see Chapter 10, “Protecting MQSeries objects” on page 103.

Under MQSeries for OS/2 Warp, you must write your own service component if you want to implement the authorization service. For example, you can create your own security features based on a third-party security product.

Name service
The name service enables the sharing of queues by allowing applications to identify remote queues as though they were local queues.

A default service component that implements the name service is provided with the MQSeries Version 5 products. It uses the Open Software Foundation (OSF) Distributed Computing Environment (DCE). You can also write your own name service component. (You might want to do this if you do not have DCE installed, for example.) By default, the name service is inactive.

For more information, see Chapter 13, “Name service” in the *MQSeries Programmable System Management* book.

User identifier service

The user identifier service is supported by MQSeries for OS/2 Warp only. It allows MQI applications in an OS/2 environment to associate a user ID (other than the default user ID, OS2) with MQSeries messages. The receiving applications are then able to identify the source of the messages. A sample user identifier service component is supplied.

Note that this is not intended to provide a ***secure*** service. There is no mechanism to prevent applications from copying this user ID.

For more information, see Appendix F, “User identifier service” on page 365.

See Chapter 11, “Installable services and components” in the *MQSeries Programmable System Management* manual for more information about the installable services.

Security

In the MQSeries Version 5 products, there are two methods of providing security:

- The Object Authority Manager (OAM) facility
- DCE security

Object Authority Manager (OAM) facility

In MQSeries for UNIX systems and MQSeries for Windows NT, 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 user groups and the OAM. A command line interface is provided to enable administrators to grant or revoke authorizations as required.

No OAM security features are provided either by MQSeries for OS/2 Warp or by OS/2 itself. You should consider what your security requirements are, and design your system to provide these facilities or, in their absence, to ensure that your applications are aware of the lack of security and are not therefore compromised.

Note: The authorization service is available in MQSeries for OS/2 Warp, but no authorization service component is supplied. If security is essential to your enterprise, consider writing your own authorization service component. This component would use the supplied interface to access the facilities provided by a third-party security manager.

For more information about creating authorization service components, see Chapter 12, “Authorization service” in the *MQSeries Programmable System Management* book.

DCE security

Channel exits that use the DCE Generic Security Service (GSS) are provided by MQSeries. For more information, see “Supplied channel-exit programs using DCE security services” in the *MQSeries Intercommunication* book.

Transactional support

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

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

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

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

For more information, see Chapter 14, “Transactional support” on page 175.

Chapter 2. An introduction to MQSeries administration

This chapter introduces the subject of MQSeries administration.

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

Local and remote administration

You administer MQSeries objects locally or remotely.

Local administration means carrying out administration tasks on any queue managers you have defined on your local system. 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.

Chapter 9, “Administering remote MQSeries objects” on page 87 describes the subject of remote administration in greater detail.

Performing administration tasks using control commands

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

See Chapter 6, “Managing queue managers using control commands” on page 49 for more information about control commands.

Performing administrative tasks using MQSC commands

You use MQSeries commands (MQSC) to manage queue manager objects, including the queue manager itself, channels, queues, and process definitions.

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

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

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

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

| MQSC commands are available on other platforms, including AS/400, and OS/390.

MQSC commands are summarized in Appendix G, “Comparing command sets” on page 367.

Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

See “Performing local administration tasks using MQSC commands” on page 60 for more information about using MQSC commands in local administration.

Performing administrative tasks using PCF commands

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

PCF commands cover the same range of functions provided by the MQSC facility.

See “PCF commands” on page 83 for more information.

| You can use the MQSeries Administration Interface (MQAI) to obtain easier
| programming access to PCF messages. This is described in greater detail in
| “Using the MQAI to simplify the use of PCFs” on page 84.

Administration on MQSeries for Windows NT

On MQSeries for Windows NT you can perform administration tasks using:

| - PCF, MQSC, and control commands
| - The MQSeries Explorer snap-in and the MQSeries Services snap-in applications running under the Microsoft Management Console (MMC)
| - The Windows NT Default Configuration application
| - MQSeries Web Administration

Using commands on MQSeries for Windows NT

You can perform administration tasks using:

- Control commands that you enter through the Windows NT command line
- The **runmqsc** control command to cause MQSC commands from standard input to be executed
- Any local or remote MQSeries application program that generates PCF commands in messages, putting them onto the command queue, SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries command server

Using the MQSeries Explorer

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

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

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

See Chapter 3, “Administration using the MQSeries Explorer” on page 29 for more information.

Using the MQSeries Services snap-in

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

- Start or stop a queue manager.
- Change the default queue manager.
- Start or stop individual MQSeries processes such as a channel initiator, or a listener.
- Start or stop the command server.
- Start or stop the service trace.
- Set a queue manager to start up automatically when you start up your workstation.

For more information, see Chapter 4, “Administration using the MQSeries Services snap-in” on page 37.

Using the Windows NT default configuration application

You can use the Windows NT Default Configuration program from the MQSeries First Steps application or the MQSeries Postcard application to create a "starter set" (or default set) of MQSeries objects which you can then administer. A summary of the default objects created is listed in Table 25 on page 346.

Using MQSeries Web Administration

MQSeries for Windows NT provides a web-based application that allows you to administer MQSeries objects on all systems in your MQSeries network from a web browser running on Windows NT, Windows 95, and Windows 98. MQSeries Web Administration shows you how to use MQSC command facilities either as individual commands or multiple commands from a script.

You start the MQSeries Web Administration server from an icon within the MQSeries Services snap-in.

For more information, see Chapter 5, "Using MQSeries Web Administration" on page 43.

Editing configuration information

All MQSeries configuration information is stored in the **Windows NT Registry**. The MQSeries configuration files, qm.ini and mqs.ini, are no longer used in MQSeries Version 5.1 There should be a simple, or close correlation between the contents of the Windows NT Registry and the MQSeries configuration files.

You edit configuration information from the MQSeries Services snap-in.

However, you may find it useful to read the descriptions of the individual attributes in the configuration files in Chapter 11, "Configuring MQSeries" on page 127. The descriptions themselves are still relevant for reference purposes.

Use the MQSeries Services snap-in to make configuration changes only. **Do not** attempt to edit the registry system file directly as this may cause your system to behave unpredictably and adversely affect the smooth running of both your MQSeries system and your Windows NT operating system.

Migrating to the Windows NT Registry

The Windows NT Registry is created when you install the operating system. When you migrate to MQSeries for Windows NT Version 5.1, the information in the configuration files you used in previous release is automatically stored in the registry by the MQSeries Services snap-in.

If you want to refer back to these files in the future, you are recommended to back them up before starting the migration process.

Viewing configuration information

You can view a description of the keys used by the Windows NT Registry from the MQSeries Information Center. You can access the MQSeries Information Center from:

- An icon in the Windows NT Start menu
- An option in the MQseries First Steps application

Understanding MQSeries file names

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

The path to a queue manager directory is formed from the following:

- A prefix, which is defined in the queue manager configuration file, qm.ini.

  In MQSeries for UNIX systems, the default prefix is:

  ```
  /var/mqm
  ```

  In MQSeries for OS/2 Warp and MQSeries for Windows NT, the default prefix is:

  ```
  c:\mqm
  ```

- A literal:

  ```
  qmgrs
  ```

- A coded queue manager name, which is the queue manager name transformed into a valid directory name. For example, the queue manager:

  ```
  queue.manager
  ```

  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:

   a. Truncate it to eight characters
   b. Append a three-character numeric suffix

For example, assuming the default prefix, the queue manager name in MQSeries for UNIX systems becomes:

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

In MQSeries for OS/2 Warp and MQSeries for Windows NT with HPFS (or NTFS), the queue manager name becomes:

```
c:\mqm\qmgrs\queue!manager
```

In MQSeries for OS/2 Warp and MQSeries for Windows NT with FAT, the queue manager name becomes:

```
c:\mqm\qmgrs\queue!ma
```

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 only a few queue manager names per machine, there can be a large number of other objects for each queue manager. Only process definitions, queues, and namelists are represented in the file system; channels are not affected by these considerations.

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

Chapter 3. Administration using the MQSeries Explorer

This information applies to MQSeries for Windows NT V5.1 only.

MQSeries for Windows NT Version 5.1 provides an administration interface called the MQSeries Explorer to perform administration tasks as an alternative to using control or MQSC commands. (Appendix G, “Comparing command sets” on page 367 shows you which operations you can perform using the MQSeries Explorer.)

The MQSeries Explorer allows you to perform remote administration of your network from a computer running Windows NT simply by pointing the MQSeries Explorer at the queue managers and clusters you are interested in. The platforms and levels of MQSeries which can be administered using the MQSeries Explorer are described in “Prerequisite software” on page 30.

The configuration steps you must perform on remote MQSeries queue managers to allow the MQSeries Explorer to administer them are outlined in “Required definitions for administration” on page 31.

This chapter contains the following topics:

What you can do with the MQSeries Explorer

With the MQSeries Explorer, you can:

- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of MQSeries objects such as queues and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel.
- View queue managers in a cluster.
- Create a new queue manager cluster using the *Create New Cluster* wizard.
- Add a queue manager to a cluster using the *Add Queue Manager to Cluster* wizard.
- Add an existing queue manager to a cluster using the *Join Cluster* wizard.

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

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

Points to consider when using the MQSeries Explorer

When deciding whether to use the MQSeries Explorer at your installation, bear the following points in mind:

- The MQSeries Explorer works best with small queue managers. If you have a large number of objects on a single queue manager you may experience delays while the MQSeries Explorer extracts the required information to present in a view. As a rough guide as to what a “large number” is, if your queue managers have more than 200 queues or 100 channels, you may want to consider using a third-party enterprise console product instead of the MQSeries Explorer.
- MQSeries clusters can potentially contain hundreds or thousands of queue managers. Because the MQSeries Explorer presents the queue managers in a cluster using a tree structure, the view can become cumbersome for large clusters. The physical size of a cluster does not affect the speed of the MQSeries Explorer dramatically because the explorer does not connect to the queue managers in the cluster until you select them.
- The message browser displays the first 200 messages on a queue. Only the first 1000 bytes of message data contained in a message are formatted and displayed on your screen. Messages containing more than 1000 bytes of message data are not displayed in their entirety.
- The MQSeries Explorer cannot administer a cluster whose repository queue managers are on MQSeries for OS/390. To avoid this problem, nominate an additional repository queue manager on a system which the MQSeries Explorer can administer. By connecting the cluster through this new repository queue manager, you can administer the queue managers in the cluster, subject to the MQSeries Explorer’s usual restrictions for supported levels of MQSeries.

Prerequisite software

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

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

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

Table 1 summarizes the platforms and command levels that support the MQSeries Explorer.

Table 1. Platforms and command levels

Platform	Command level
AIX and UNIX variants	Command level 221 and above
OS/400®	Command level 320 and above
OS/2 and Windows NT	Command level 201 and above
VMS and Tandem	Command level 221 and above

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

Required definitions for administration

Ensure that you have satisfied the following requirements before attempting to use the MQSeries Explorer. Check that:

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

   You can create the channel using the following MQSC command:

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

   The command cited creates a very basic channel definition. If you want a more sophisticated definition, to set up security, for example, additional parameters are required.

Showing and hiding queue managers and clusters

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

To show a remote queue manager, you must select the *Show a remote queue manager* radio button and type in the name of the remote queue manager and the connection name in the field provided. The connection name is the IP address, or host name, of the machine you are trying to connect to, with an optional port

number. This connection name is used to establish a client connection to the remote queue manager using its SYSTEM.ADMIN.SVRCONN server connection channel.

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

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

Cluster membership

The MQSeries Explorer needs to maintain up to date administration data about clusters in order to be able to communicate effectively with them and to display correct cluster information when requested to do so. In order to do this, the MQSeries Explorer needs the following information from you:

- The name of a repository queue manager
- The connection name of the repository queue manager if it is on a remote queue manager

With this information, the MQSeries Explorer can:

- Use the repository queue manager to obtain a list of queue managers in the cluster.
- Administer the queue managers that are members of the cluster and are on supported platforms and command levels.

Administration is not possible if:

- The chosen repository becomes unavailable. The MQSeries Explorer does **not** switch to an alternative repository.
- The chosen repository cannot be contacted over TCP/IP.
- The chosen repository is running on a queue manager that is running on a platform and command level not supported by the MQSeries Explorer.

The cluster members that can be administered can be local, or they can be remote if they can be contacted using TCP/IP. The MQSeries Explorer connects to local queue managers that are members of a cluster directly, without using a client connection.

Security

If you are using MQSeries in an environment where it is important for you to control user access to particular objects, you may need to consider the security aspects of using the MQSeries Explorer.

Authorization to run the MQSeries Explorer

Before the MQSeries Explorer is enabled, you must:

- Ensure that chosen users have the correct level of authorization. This means being one of the following:
  - A member of the mqm group
  - A member of the administrator group on the machine running the MQSeries Explorer
  - Logged on using the SYSTEM ID

Group membership at logon time is used for authorization purposes, if membership is changed so that a user can access the MQSeries Explorer, that user must log off and log back on again.

Furthermore, some operations may require you to have authorization to use individual objects or object types. The MQSeries Explorer uses existing MQSeries rules for security to ensure that this happens. For example, you must have display authority for a queue to be able to view its attributes in the MQSeries Explorer.

Security for connecting to remote queue managers

The MQSeries Explorer connects to remote queue managers as an MQI client application. This means that each remote queue manager must have a definition of a server connection channel and a suitable TCP/IP listener. If you do not specify a nonblank value for the MCAUSER attribute of the channel, or use a security exit, it is possible for a malicious application to connect to the same server connection channel and gain access to the queue manager objects with unlimited authority.

The default value of the MCAUSER attribute is a blank. If you specify a nonblank user name as the MCAUSER attribute of the server connection channel, all programs connecting to the queue manager using this channel run with the identity of the named user and have the same level of authority.

Using a security exit

A more flexible approach is available by installing a security exit on the server connection channel SYSTEM.ADMIN.SVRCONN on each queue manager which is to remotely administered.

A matching security exit can be installed on the machine on which the MQSeries Explorer is being used. This allows complete flexibility in the authentication processing performed when a client connection is established to give consistency with your enterprise's MQSeries security policy.

Enabling a security exit

The *IBM MQSeries* node Property Sheet allows you to specify the name of a security exit and provide security exit data to be used for all client connections to remote queue managers that are established by the MQSeries Explorer. This Property Sheet is also displayed when the MQSeries Explorer is added to a console using the MMC *Add/Remove Snap-in* dialog box.

If you change this information after client connections have been established, the new information affects only those connections set up after the change has been made. Existing connections are **not** broken and re-established.

Data conversion

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

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

An error message is issued if you try to establish a connection between the MQSeries Explorer and a queue manager whose CCSID the MQSeries Explorer does not recognize.

Supported conversions are described in Appendix F, "Code page conversion tables" in the *MQSeries Application Programming Reference* manual.

Saving and loading console files

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

The following information is saved in a .MSC file:

- Details of the queue managers and clusters showing in the console. The names of the queue managers that are members of the visible clusters are not saved.
- The security exit name and the security exit data for client connections to remote queue managers.
- Whether the MQSeries Explorer automatically loads the local queue managers and the clusters of which they are members when the .MSC file is loaded.
- Any non-default configuration of columns visible in each view.
- Filtering options for the objects visible in each view.

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

Switching off the automatic population facility

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

The default behavior is to automatically determine:

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

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

Chapter 4. Administration using the MQSeries Services snap-in

This information applies to MQSeries for Windows NT V5.1 only.

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

The MQSeries Services snap-in is an administration tool that should only be used by experienced staff who are authorized to make changes to MQSeries objects and services.

This chapter contains the following:

What you can do with the MQSeries Services snap-in

All the functions offered by the MQSeries Services snap-in can be used to administer local or remote MQSeries for Windows NT servers, except for the *Alert monitor* function which records and notifies you of problems in your MQSeries system. This function can be used on your local system only. See "Using the MQSeries alert monitor application" on page 38 for more information.

With the MQSeries Services snap-in, you can:

- Start or stop a queue manager (on your local machine or on remote NT machines).
- Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- Create and delete queue managers, command servers, channel initiators, trigger monitors, and listeners.
- Set any of the services to start up automatically or manually during system start up.
- Modify the properties of queue managers. *This function replaces the use of stanzas in configuration (mqs.ini and qm.ini) files*.
- Change the default queue manager.
- Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- Modify the behavior of MQSeries if a particular service fails, for example, retry starting the service x number of times.
- Start or stop the service trace.

- Start or stop MQSeries Web Administration. (For more information, see Chapter 5, “Using MQSeries Web Administration” on page 43.)

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

Prerequisite software

Before you can use the MQSeries Services snap-in, you must have the following software installed on your computer:

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

Using the MQSeries Services snap-in

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

Green Healthy; no alerts at present

Blue Indeterminate; MQSeries is starting up or shutting down

Yellow Alert; one or more services are failing or have already failed

When you click on the icon with your right mouse button, a context menu appears. From this menu, select the `MQSeries Services` option to bring up the MMC. The MQSeries Services snap-in is already loaded and is ready to use.

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

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

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

Using the MQSeries alert monitor application

The MQSeries alert monitor is an error detection tool that identifies and records problems with MQSeries on a local machine.

The MQSeries alert monitor displays information about the current status of the local installation of an MQSeries server.

From the MQSeries alert monitor, you can:

- Access the MQSeries Services snap-in directly
- View information relating to all outstanding alerts
- Shut down the IBM MQSeries service on the local machine
- Route alert messages over the network to a configurable user account, or to an NT workstation or server

Looking at MQSeries alert monitor information

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

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

MQSeries Services snap-in recovery facilities

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

- Restart the queue managers.
- Execute a program. You may like to set up a paging to notify you of a failure, or have electronic mail sent, for example.
- Restart the server.
- Log the fact that a failure has occurred, but take no action.

Security

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

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

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

Because AMQMSRVN must be shared between non-interactive and interactive logon sessions, it is launched under a special user account. This special user account is called “MQAdmin” and is created when MQSeries is installed.

The password for MQAdmin is generated at installation time and is used only during the installation process to create the account itself and to configure the

logon environment for AMQMSRVN. The password is not known outside this "one-time" processing and is stored by the Windows NT operating system in a secure part of the Windows NT Registry.

When the service is running, AMQMSRVN is launched and remains running for as long as the service is running. An MQSeries administrator who logs onto the server after AMQMSRVN is launched can use the MQSeries Services snap-in to administer queue managers on the server. By doing this, the MQSeries administrator causes the MQSeries Services snap-in to connect to the existing AMQMSRVN process. These two actions need different levels of permission before they can work:

- The launch process requires a launch permission.
- The MQSeries administrator requires "Access" permission.

Controlling access

When IBM MQSeries is installed, default access permissions are set up for the AMQMSRVN process. These default access permissions grant the following access permissions:

- mqm (local MQ Administrators Group)
- SYSTEM (local account that the IBM MQSeries service runs under)
- Administrators (local administrators of this machine)

These permissions restrict access to the alert monitor task bar application and the MQSeries Services snap-in and MQSeries Explorer snap-in to members of the mqm group alone. Other users attempting to access these functions are denied access.

Before you can grant or deny users access to the MQSeries Services snap-in, you must configure the access permissions of the objects involved. A tool called *DCOMCNFG.EXE* which is shipped with Windows NT can be used to do this.

Using the DCOMCFNG.EXE tool

To start DCOMCNFG.EXE:

- Click on the Windows Start button
- Select "Run"
- Type "dcomcnfg" in the open input field
- Click OK

A list of applications is displayed. From this list:

- Find and highlight the "IBM MQSeries Services" entry.
- Click on the Properties button. This displays information about the location of the DCOM server (AMQMSRVN.EXE) together with its identity and security properties.
- Select the "Security" page to view or modify the launch, access or configuration permissions.
- Stop the IBM MQSeries service from the Windows NT control panel and restart it for your changes to take effect. (If your changes affect a user who is currently logged on, that user must log off and log back on again.)

In addition to being able to add to the list of users that are allowed access to a service, you can deny access to specific users and groups. This means that you to grant access to a group of users (by specifying a group name) but deny access to individuals within that group.

Controlling remote access

You can also grant or deny access to users of remote machines using the DCOMCNFG.EXE tool.

The DCOM server can be turned on or off for the entire server using the appropriate setting on the Default Properties page.

Changing the MQAdmin user account

To change the password that has been generated for the MQAdmin user account:

- Stop the IBM MQSeries service.
- Close any MQSeries programs that are using the AMQMSRVN COM server (this includes snap-ins, alert monitor, task bar and so on).
- Use the *User Manager* to change the MQAdmin password in the same way that an individual's password would be changed. The User Manager is a Windows NT system management tool which allows system administrators to add, delete, or change users on an MQSeries system.
- Use the DCOMCNFG.EXE tool to bring up the properties pages for the IBM MQSeries service.
- Select the Identity Page.
- Modify the password given for the MQAdmin user account.

Because the AMQMSRVN COM server runs under the MQAdmin user account, it is this account that executes any MQSeries commands that are issued by user interface applications, or performed automatically on system startup, shutdown, or service recovery. Therefore the MQAdmin user account must have MQSeries administration rights. By default it is added to the local mqm group on the server. If this membership is removed, the MQSeries service fails to work properly.

If a security problem arises with the DCOM configuration or with the MQAdmin user account, error messages and descriptions appear in the system event log. One common error is for a user not to have access or launch rights to the server. This error appears in the system log as a DCOM error with the following message description:

```
Access denied attempting to launch a DCOM server.  The server is:
{55B99860-F95E-11d1-ABB6-0004ACF79B59}
```

Chapter 5. Using MQSeries Web Administration

This information applies to MQSeries for Windows NT V5.1 only.

MQSeries for Windows NT Version 5.1 provides an administration interface called *MQSeries Web Administration*. This allows you to perform the following tasks using a Java-enabled browser such as Netscape Navigator or Microsoft Internet Explorer:

- Log on as an MQSeries Administrator
- Select a queue manager and issue MQSC commands against it
- Create, edit, and delete MQSC scripts

Appendix G, "Comparing command sets" on page 367 shows you which operations you can perform using MQSeries Web Administration.

This chapter contains the following topics:

Points to consider when using MQSeries Web Administration

When deciding whether or not to use MQSeries Web Administration at your installation, bear the following points in mind:

- The MQSeries Web Administration web server requires a dedicated IP port number.
- MQSeries Web Administration can be accessed from the Internet if permitted to do so by your network configuration.
- All users of MQSeries Web Administration require an active Windows NT user ID on the server computer with sufficient user rights to run MQSC commands. See "Authorization to run MQSeries Web Administration" on page 46 for more information.
- To administer queue managers on computers other than the one running the MQSeries Web Administration web server, MQSeries message channels must be configured between the systems.

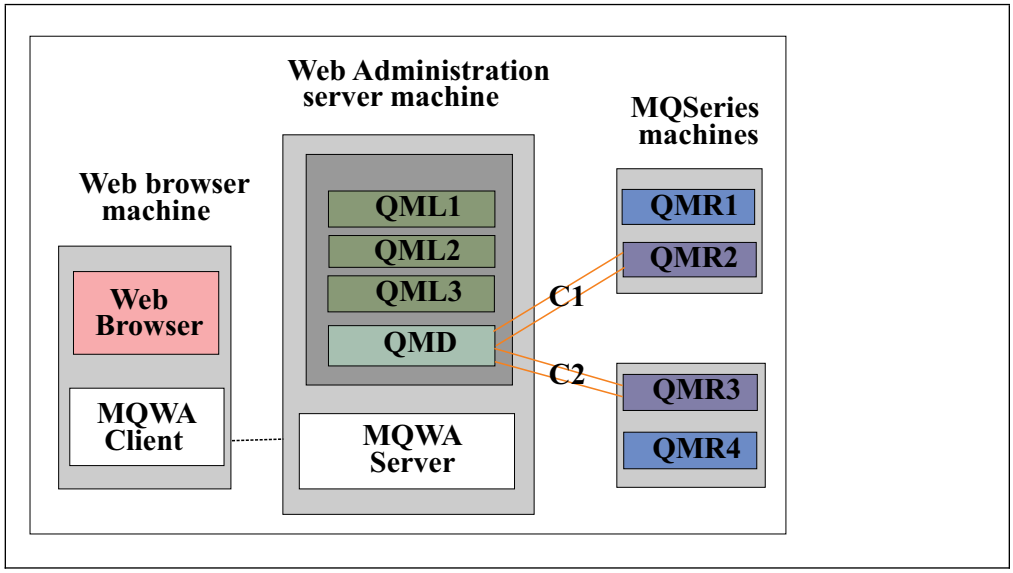


Figure 1. MQSeries Web Administration

Figure 1 shows a MQSeries Web Administration client which can administer the following queue managers when the administrator is logged into the MQSeries Web Administration server. (Note that MQWA is an abbreviation for MQSeries Web Administration.)

The local queue managers QML1, QML2, QML3, and QMD reside on the MQSeries Web Administration server machine. The remote queue managers QMR2 and QMR3 are connected to the default queue manager, QMD, via the message channels C1 and C2. Note that the remote queue managers QMR1 and QMR4 cannot be administered from this MQSeries Web Administration client because there are no channels defined to connect them to the default queue manager, QMD.

Prerequisite software

Before you can use MQSeries Web Administration, you must have the following installed on your computers.

Prerequisite software for the server side

- Windows NT Version 4 (SP3)
- Microsoft Internet Explorer Version 4.01 (SP1)
- MQSeries for Windows NT Version 5.1 server and Web Administration

Prerequisite software for the client side

Install either of the following browsers:

- Netscape Navigator 4.04 with the Java AWT upgrade
- Microsoft Internet Explorer Version 4.01 (SP1)

Encryption policies

Two levels of encryption are used for administering MQSeries objects over the web. They are:

- **Authentication during log on**
  The RSA Public Key Encryption Algorithm (PKA) with a 512-bit key is used for the initial client (web browser) logon to the server. The server authenticates the client, and a separate 40-bit key is created for the actual administration tasks described in the rest of this chapter. Consequently 512-bit RSA PKA is the level of encryption applied to the user's Windows NT user name and password.

  A new RSA public and private key pair is generated each time the MQSeries Web Administration server is started.

- **Authentication when performing MQSeries Web Administration**
  The RC4 encryption algorithm using 40-bit keys is used for all other data flowing between client and server.

Starting up MQSeries Web Administration server

The MQSeries Web Administration server can be started and stopped from an icon in the MQSeries Services snap-in. The server is configured to start automatically when installed.

Logging on as an MQSeries administrator (client side)

Connect your web browser to the MQSeries Web Administration web server using a URL of the form:

```
http://<hostname> : <port_number>
```

where:

<`hostname`**>** is the IP host name of the computer running the web server.

<`port_number`**>**
is the IP port number assigned to the web server. The default value for `port_number` is 8081. However, this value may be changed using the MQSeries Services snap-in.

This URL must be made known to all MQSeries administrators who will be using MQSeries Web Administration.

The left-hand pane of the browser window contains a navigation area. To log on as an MQSeries Web Administrator and administer MQSeries objects:

- Select the "Logon" option.
- Use the Logon panel to enter your Windows NT user ID and password for MQSeries Web Administration.
- Click Enter to begin the logging on process.

Authorization to run MQSeries Web Administration

Your user ID needs the necessary administration privileges on the MQSeries server to perform administration tasks. Therefore, before attempting to log on to MQSeries Web Administration, ensure that you have the correct level of authorization. This means being one or more of the following:

- A member of the mqm group
- A member of the administrator group on the machine running MQSeries Web Administration
- Logged on using the SYSTEM ID

Some operations may require you to have authorization to use individual objects or object types. MQSeries Web Administration uses existing MQSeries rules for security to ensure that this happens.

Security for connecting to remote queue managers

MQSeries Web Administration connects to remote queue managers using MQSC. The web administration server adopts the user ID of each logged-on administrator prior to invoking MQSC commands on the administrator's behalf. Therefore, administrators have exactly the same privileges from MQSeries Web Administration as they would have using the **runmqsc** command locally on the web administration server.

Administering queue managers

Use the *Administration* panel to select the queue manager that you want to work with and to run MQSC commands or command scripts against.

Detailed information about performing these functions can be found in the online help for MQSeries Web Administration.

Administering local queue managers

To administer a local queue manager, either select the name from the drop-down list or enter the name directly in the `queue manager name` field.

Administering remote queue managers

To administer a remote queue manager, enter the queue manager name in the `queue manager name` field.

Before you can administer a remote queue manager through MQSeries Web Administration, you must define a channel between the local default queue manager and the remote queue manager.

For more information about defining channels for remote administration, see Chapter 9, "Administering remote MQSeries objects" on page 87.

Using MQSeries command scripts

MQSeries command scripts are files which a contain a sequence of MQSeries commands to be executed to perform a specific task. Script file names can be up to 251 characters in length, and can contain any alphanumeric characters other than \, /, :, *, ?, ", <, >, or |. Names longer than 251 characters are truncated. They have a suffix of *.mqs* and are stored in a folder on the MQSeries Web Administration server.

MQSeries Web Administration provides the following to assist with the creation and running of scripts:

- A scripting language to simplify the writing of scripts which can be run on any platform supported by MQSeries Web Administration
- A script management tool to help you with the creation, editing, and deletion of scripts
- A user interface for the selection and running of scripts

You use the *Script Management* panel to create and manage command scripts (create, edit, and delete).

You can edit an existing script or create a new one. Scripts are saved by MQSeries Web Administration in the public scripts directory if you have created a public script[1], or in a private scripts directory if you have created a private script[2].

To delete a script from the server, you must open the script in the edit area, and, when the file is displayed, delete it using the `Delete` option.

Detailed information about performing these functions can be found in the online help for MQSeries Web Administration.

1 A public script is a script that is available for use by all authorized administrators.

2 A private script is a script that is available for use by a specific administrator user ID only.

Configuring the MQSeries Web Administration server

There are several options that you can use to configure the MQSeries Web Administration server, as shown in Table 2. You do this from the relevant property pages in the MQSeries Services snap-in.

Table 2. Configuration options for MQSeries Web Administration

Value	Type	Default value	Description
MaxClients	REG_DWORD	50	The maximum number of administrators that can use MQSeries Web Administration simultaneously.
RemoteQMTimeout	REG_DWORD	30	The number of seconds before an attempt to access a remote queue manager times out.
SessionTimeout	REG_DWORD	60	The number of minutes of inactivity before an administrators session times out (that is, will close).
Trace	REG_SZ	Yes	Enable or disable a graphical trace window(1) (MQSeries trace is always enabled).
WebPort	REG_DWORD	8081	The IP port used by the web server.

Note for Table 2:

1. The graphical trace window is a window on the screen that contains a copy of most of the information that is logged to MQSeries trace.

   Because enabling this window can have a detrimental effect on the performance of the MQSeries Web Administration server, you should only use it on the recommendation of your IBM service representative.

Chapter 6. Managing queue managers using control commands

This chapter describes how you can perform operations on queue managers and command servers using control commands.

It contains the following sections:

Using control commands

You use control commands to perform operations on queue managers, command servers, and channels. Control commands can be divided into three categories, as shown in Table 3.

Table 3. Categories of control commands

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

For information about administration tasks for channels, see Chapter 5, "DQM implementation" in the *MQSeries Intercommunication* book.

Using control commands (MQSeries for UNIX systems)

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

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

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

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

Using control commands (MQSeries for OS/2 Warp and MQSeries for Windows NT)

In MQSeries for OS/2 Warp and MQSeries for Windows NT, you enter control commands at a command prompt. In these environments, control commands and their flags are not case sensitive, but arguments to those commands (such as queue names and queue-manager names) are case sensitive.

For example, in the command:

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

- The command name can be entered in uppercase or lowercase, or a mixture of the two. These are all valid: `crtmqm`, `CRTMQM`, and `CRTmqm`.
- The flag can be entered as `-u`, `-U`, `/u`, or `/U`.
- The arguments `SYSTEM.DEAD.LETTER.QUEUE` and `jupiter.queue.manager` must be entered exactly as shown.

Chapter 17, “MQSeries control commands” on page 279 describes the syntax and purpose of each command.

Using the MQSeries Explorer (MQSeries for Windows NT only)

For MQSeries for Windows NT only, you can use the MQSeries Explorer to perform the operations described in this chapter, except for:

- Making an existing queue manager the default
- Preemptive shutdown

The tables published in Appendix G, “Comparing command sets” on page 367 summarize which control commands have an equivalent MQSeries Explorer implementation.

Creating a queue manager

A queue manager manages the resources associated with it, in particular the queues that it owns. It provides queuing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete MQSeries objects.

Before you can do anything with messages and queues, you must create at least one queue manager and its associated objects. To create a queue manager, you use the MQSeries control command **crtmqm**. The **crtmqm** command automatically creates 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. When a queue manager and its objects have been created, you use the **strmqm** command to start the queue manager.

Guidelines for creating queue managers

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

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

  One way of ensuring uniqueness is to prefix each queue manager name with its own unique node name. For example, if a node is called `accounts`, you 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 communication 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 are allowed to create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. However, such complications are best avoided.

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

  In production systems, many nodes will be run with a single queue manager, but larger server machines may run with multiple queue managers.

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

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

  If you decide to change the default queue manager, be aware that 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* you have changed the default queue manager will 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.

  To create a default queue manager, specify 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.

  For a detailed description of this command and its parameters, see “crtmqm (Create queue manager)” on page 284.

- **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.

  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” on page 65 for an example of an MQSC ALTER command.

  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 12, “The MQSeries dead-letter queue handler” on page 157 for further information about the dead-letter queue handler.

- **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 manage, 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" on page 70 for more information.

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

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

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

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

  For more information about specifying logging parameters:

  – Using the **crtmqm** command, see "crtmqm (Create queue manager)" on page 284.

  – Using configuration files, see "The LogDefaults stanza" on page 132, and "The Log stanza" on page 138.

Backing up configuration files after creating a queue manager

There are two types of configuration file:

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.

After creating a queue manager, you are recommended to back up your configuration files.

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

For more information about configuration files, see Chapter 11, "Configuring MQSeries" on page 127.

| If you use MQSeries for Windows NT Version 5.1, or later, configuration information
| is stored in the Windows NT Registry and not in configuration files. You use the
| MQSeries Services snap-in to make changes to the Windows NT Registry.

Creating a default queue manager

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

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

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

where:

-q	Indicates that this queue manager is the default queue manager.
-d MY.DEFAULT.XMIT.QUEUE	Is the name of the default transmission queue.
-u SYSTEM.DEAD.LETTER.QUEUE	Is the name of the dead-letter queue.
saturn.queue.manager	Is the name of this queue manager. This must be the last parameter specified on the **crtmqm** command.

The system and default objects are listed in Appendix A, “System and default objects” on page 343.

For MQSeries for UNIX systems only

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

Starting a queue manager

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

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

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

Starting a queue manager automatically

In MQSeries for Windows NT only, a queue manager can be invoked automatically when the system starts using the MQSeries Services snap-in. See Chapter 4, “Administration using the MQSeries Services snap-in” on page 37 for more information.

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 `Name` attribute of 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 the `DefaultQueueManager` stanza:

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

See Chapter 11, “Configuring MQSeries” on page 127 for information about configuration files.

If you use MQSeries for Windows NT Version 5.1, or later, use the MQSeries Services snap-in to make changes to configuration information for Windows NT queue managers in the Windows NT Registry. See Chapter 4, “Administration using the MQSeries Services snap-in” on page 37 for more information.

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

Stopping a queue manager

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

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

Quiesced shutdown

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

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

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

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

However, if you issue:

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

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

Immediate shutdown

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

Use this as the normal way to stop the queue manager, optionally after a quiesce period. For an immediate shutdown, type:

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

Preemptive shutdown

Attention!

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

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

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

This stops all queue manager code immediately.

If this method still does not work, see “Stopping a queue manager manually” on page 359 for an alternative solution.

For a detailed description of the endmqm command and its options, see “endmqm (End queue manager)” on page 306.

If you have problems shutting down a queue manager

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

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

If a problem does occur while stopping the queue manager, break out of the **endmqm** command using Ctrl-C.

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

Restarting a queue manager

To restart a queue manager, type:

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

Deleting a queue manager

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

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

Notes:

1. Deleting a queue manager is a drastic step, because you also delete all resources associated with that queue manager, including all queues and their messages, and all object definitions.
2. In MQSeries for Windows NT, the **dltmqm** command also removes a queue manager from the automatic start-up list (described in “Starting a queue manager automatically” on page 55).

For a description of the **dltmqm** command and its options, see “dltmqm (Delete queue manager)” on page 289. 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” on page 360 for an alternative.

Chapter 7. Administering local MQSeries objects

This chapter describes how to administer local MQSeries objects to support application 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

MQSeries application programs need certain objects before they can run successfully. For example, Figure 2 shows an application that removes messages from a queue, processes them, and then sends some results to another queue on the same queue manager.

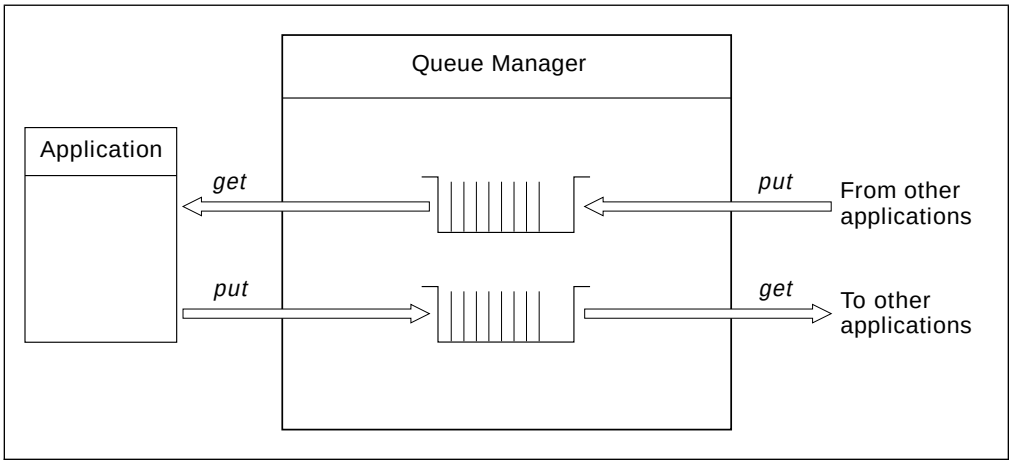


Figure 2. Queues, messages, and applications

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

Before this application can be run, the following conditions must be satisfied:

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

- The second queue, on which the application puts the messages, must also be defined.
- The application must be able to connect to the queue manager. To do this it must be linked to the product code. See Chapter 7, “Connecting and disconnecting a queue manager” in 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 2 on page 59.

Performing local administration tasks using MQSC commands

In this section, we assume that you will be issuing commands using the **runmqsc** command. It gives you a general but detailed introduction to MQSC commands and shows you how to use them on some commonly performed tasks. However, if you use MQSeries for Windows NT Version 5.1 or later, you can perform the operations described in this section using the MQSeries Explorer or MQSeries Web Administration. See Chapter 3, “Administration using the MQSeries Explorer” on page 29 and Chapter 5, “Using MQSeries Web Administration” on page 43 for more information.

You can use MQSeries script commands (MQSC) to manage queue manager objects, including the queue manager itself, clusters, channels, queues, namelists, and process definitions. This section deals with queue managers, queues and process definitions; for information about administering channel objects, see Chapter 5, “DQM implementation” in the *MQSeries Intercommunication* book.

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

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

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

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

MQSC commands are available on other platforms, including AS/400 and OS/390.

MQSC commands are summarized in Appendix G, “Comparing command sets” on page 367.

Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

MQSeries object names

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

When you 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 uppercase or lowercase. Object names in MQSC commands are folded (that is, QUEUE and queue are not differentiated), unless the names are within single quotation marks. If quotation marks are not used, the object is processed with a name in uppercase. See “Rules for naming MQSeries objects” in the *MQSeries Command Reference* manual for more information.

However, the **runmqsc** command invocation, in common with all MQSeries control commands, is case sensitive in some MQSeries environments. See “Using control commands” on page 49 for more information.

Standard input and output

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

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

Using the MQSC facility interactively

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

```
runmqsc
```

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)
```

Continuation characters must used to indicate that a command is continued on the following line:

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

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

Feedback from MQSC commands

When you issue 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
AMQ8426: Valid MQSC commands are:


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

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 Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual for the correct syntax.

Ending interactive input to MQSC

To end interactive input of MQSC commands, enter the MQSC END command:

```
END
```

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

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
```

Typical output from this command is shown in Figure 3 on page 64.

```
     1 : display qmgr all
AMQ8408: Display Queue Manager details.
   DESCR( )                                DEADQ( )
   DEFXMITQ( )                             CHADEXIT( )
   CLWLEXIT( )                             CLWLDATA( )
   REPOS( )                                REPOSNL( )
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)    QMNAME(saturn.queue.manager)
   CRDATE(1998-09-25)                      CRTIME(09.40.06)
   ALTDATE(1998-09-25)                     ALTTIME(09.40.06)
   QMID(saturn.queue.manager_1998-09-25_09.40.06)
   TRIGINT(999999999)                      MAXHANDS(256)
   MAXUMSGS(10000)                         AUTHOREV(DISABLED)
   INHIBTEV(DISABLED)                      LOCALEV(DISABLED)
   REMOTEEV(DISABLED)                      PERFMEV(DISABLED)
   STRSTPEV(ENABLED)                       CHAD(DISABLED)
   CHADEV(DISABLED)                        CLWLLEN(100)
   MAXMSGL(4194304)                        CCSID(850)
   MAXPRTY(9)                              CMDLEVEL(510)
   PLATFORM(WINDOWSNT)                     SYNCPT
   DISTL(YES)

   2 : display qmgr
AMQ8408: Display Queue Manager details.
   DESCR( )                                DEADQ( )
   DEFXMITQ( )                             CHADEXIT( )
   CLWLEXIT( )                             CLWLDATA( )
   REPOS( )                                REPOSNL( )
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)    QMNAME(saturn.queue.manager)
   CRDATE(1998-09-25)                      CRTIME(09.40.06)
   ALTDATE(1998-09-25)                     ALTTIME(09.40.06)
   QMID(saturn.queue.manager_1998-09-25_09.40.06)
   TRIGINT(999999999)                      MAXHANDS(256)
   MAXUMSGS(10000)                         AUTHOREV(DISABLED)
   INHIBTEV(DISABLED)                      LOCALEV(DISABLED)
   REMOTEEV(DISABLED)                      PERFMEV(DISABLED)
   STRSTPEV(ENABLED)                       CHAD(DISABLED)
   CHADEV(DISABLED)                        CLWLLEN(100)
   MAXMSGL(4194304)                        CCSID(850)
   MAXPRTY(9)                              CMDLEVEL(510)
   PLATFORM(WINDOWSNT)                     SYNCPT
   DISTL(YES)
```

Figure 3. Typical output from a DISPLAY QMGR command

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).

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

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

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

Using a queue manager that is not the default

To run MQSC commands on a local queue manager other than the default queue manager, you specify the name of the queue manager on input to the **runmqsc** command. For example, to run MQSC commands on queue manager `jupiter.queue.manager`, use the command:

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

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

You can also run MQSC commands on a remote queue manager; see "Issuing MQSC commands remotely" on page 93.

Altering queue manager attributes

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

```
runmqsc jupiter.queue.manager

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

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

Running MQSC commands from text files

Running MQSC commands interactively is suitable for quick tests, but if you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a text file. (See "Standard input and output" on page 61 for information about `stdin` and `stdout`.) To do this, first create a text file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. For example, the following command runs a sequence of commands contained in the text file `myprog.in`:

```
runmqsc < myprog.in
```

Similarly, you can also redirect the output to a file. A file containing the MQSC commands for input is called an *MQSC command file*. The output file containing replies from the queue manager is called the *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`. Figure 4 shows an extract from the MQSC command file `myprog.in` and Figure 5 on page 67 shows the corresponding extract of the output in `myprog.out`.

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

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

MQSC command files

MQSC commands are written in human-readable form, that is, in ASCII text. Figure 4 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

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

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

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

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 Figure 5. 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 attempts to process only those commands that have no syntax errors.

```
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.
      :
 .
 .
```

Figure 5. Extract from the MQSC report file, myprog.out

Running the supplied MQSC command files

These MQSC command files are supplied with MQSeries:

amqscos0.tst	Definitions of objects used by sample programs.
amqscic0.tst	Definitions of queues for CICS transactions.

In MQSeries for UNIX systems, these files are located in the directory **mqmtop**/samp; see “The base directory” on page xv for details of the installation directory **mqmtop**.

In MQSeries for OS/2 Warp and MQSeries for Windows NT, these files are located in the directory c:\mqm\tools\mqsc\samples.

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 Figure 5 on page 67.

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

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

the -w flag, which you use to indicate that the queue manager is remote, is ignored, and the command is run locally in verification mode.

Resolving problems with MQSC

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

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

- Use the indirection operator < when redirecting input from a file. If you omit the indirection operator, the queue manager interprets the file name as a queue manager name, and issues the following error message:

```
AMQ8118: MQSeries queue manager does not exist.
```

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

To do this, look in the MQSeries configuration file, mqs.ini. This file contains the names of the queue managers and the name of the default queue manager, if you have one.

- The queue manager should already be started, if it is not, start it; see "Starting a queue manager" on page 54. 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:

```
AMQ8146: MQSeries queue manager not available.
```

To correct this type of problem, see "Making an existing queue manager the default" on page 55.

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

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

- You cannot enter MQSC commands before you issue the **runmqsc** command.
- You cannot run control commands from **runmqsc**. For example, you cannot issue the **strmqm** command to start a queue manager while you are running MQSC interactively.

```
runmqsc
 .
 .
Starting MQSeries Commands.

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

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

See also “If you have problems using MQSC remotely” on page 95.

Working with local queues

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

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues 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” on page 71.
2. USAGE (NORMAL) indicates that this queue is not a transmission queue.
3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute, if you want to overwrite the existing definition of a queue, but see also “Changing local queue attributes” on page 72.

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. This queue is automatically created when you create the queue manager. You can modify this definition if required. There is no need to rename it, although you can if you like.

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

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

MQSeries provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see Chapter 12, “The MQSeries dead-letter queue handler” on page 157.

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)
```

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(5000)
   MAXMSGL(4194304)                        CURDEPTH(0)
     5 : end
```

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

Copying a local queue definition

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

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

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

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” on page 70, 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.

Browsing queues

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

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

Source	**mqmtop**/samp/amqsbcg0.c
Executable	**mqmtop**/samp/bin/amqsbcg

In MQSeries for OS/2 Warp and MQSeries for Windows NT, the default file names and paths are:

Source	c:\mqm\tools\c\samples\amqsbcg0.c
Executable	c:\mqm\tools\c\samples\bin\amqsbcg.exe

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

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

Typical results from this command are shown in Figure 6 on page 75.

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

 MQOPEN - 'SYSTEM.ADMIN.QMGR.EVENT'


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

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

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

Figure 6 (Part 1 of 2). Typical results from queue browser

```
****   Message      ****

 length - 104 bytes

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



 No more messages
 MQCLOSE
 MQDISC
```

Figure 6 (Part 2 of 2). Typical results from queue browser

Monitoring local queues with the Windows NT Performance Monitor

Administrators of MQSeries for Windows NT can monitor the performance of local queues using the *Windows NT Performance Monitor*.

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

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

For each queue, you can view information relating to the following:

- The current queue depth
- The queue depth as a percentage of the maximum queue depth
- The number of messages being placed on the queue per second
- The number of messages being removed from the queue per second

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

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

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

The performance of queues on computers other than that on which the Performance Monitor is running can be monitored. To monitor the queues on

another computer, select your target computer from the Performance Monitor, which works using the Windows Network Neighborhood hierarchy.

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 run time.

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

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

Defining an alias queue

The following command creates an alias queue:

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

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

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

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

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

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

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

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 these attributes with local queues.

Using other commands with alias queues

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's attributes

DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE)


* 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 Chapter 2, “The MQSeries commands” in 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 (PERMDYN)
```

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

DISPLAY QUEUE (GREEN.MODEL.QUEUE)


* 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 Chapter 14, “Starting MQSeries applications using triggers” in the *MQSeries Application Programming Guide*.

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

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *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)
```

where:

`QLOCAL (MOTOR.INSURANCE.QUEUE)`
: Specifies the name of the application queue being defined.

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

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

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

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

TRIGGER
: Is the trigger attribute value.

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

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

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

Defining an initiation queue

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

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

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') +
               USERDATA ('open, close, 235')
```

Where:

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

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

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

`APPLICID ('/u/admin/test/IRMP01')`
: Is the name of the application executable file, specified as a fully qualified file name. In MQSeries for OS/2 Warp and Windows NT, a typical `APPLICID` value would be `c:\appl\test\irmp01.exe`.

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

Displaying your process definition

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

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


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

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

Chapter 8. Automating administration tasks

This chapter assumes that you have experience of administering MQSeries objects.

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

This chapter describes:

- How to use programmable command formats to automate administration tasks in "PCF commands," which includes a description of support for Microsoft's Active Directory Service Interfaces (ADSI).
- How to use the command server in "Managing the command server for remote administration" on page 85

PCF commands

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

PCF commands cover the same range of functions provided by the MQSC facility.

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

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

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

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

Message descriptor
This is a standard MQSeries message descriptor, in which:

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

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

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

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

For a complete description of the PCF data structures and how to implement them, see "PCF command messages" in the *MQSeries Programmable System Management* manual.

Attributes in MQSC and PCFs

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

Object attributes in PCF, which are not limited to eight characters, are shown in this book in italics. For example, the PCF equivalent of RQMNAME is *RemoteQMgrName*.

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see "Escape" in the *MQSeries Programmable System Management* manual.

Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to MQSeries that is available on the AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT platforms.

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

The MQAI can be used:

- **To simplify the use of PCF messages**
  The MQAI is an easy way to administer MQSeries; you do not have to write your own PCF messages and this avoids the problems associated with complex data structures.

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

  On the other hand, programs written using the MQAI pass parameters into the appropriate data bag and only one statement is required for each structure. The use of MQAI data bags removes the need for you to handle arrays and allocate storage, and provides some degree of isolation from the details of the PCF.
- **To implement self-administering applications and administration tools**
  For example, the Active Directory Services provided by MQSeries for Windows NT Version 5.1 uses the MQAI.
- **To handle error conditions more easily**
  It is difficult to get return codes back from MQSC commands, but the MQAI makes it easier for the program to handle error conditions.

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

For more information about using the MQAI, see the *MQSeries Administration Interface Programming Guide and Reference* book.

For more information about PCFs in general, see Chapter 7, “Using Programmable Command Formats” in the *MQSeries Programmable System Management* book.

Active Directory Services

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

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

IBM MQSeries support for the ADSI is implemented through the use of the **IBMMQSeries namespace**.

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

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

`www.microsoft.com`

For more information about Component Object Model (COM) interfaces, see the *MQSeries for Windows NT Using the Component Object Model Interface* book.

Note: To access a queue manager, it must be running and have an associated command server.

Managing the command server for remote administration

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

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

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

There are separate control commands for starting and stopping the command server. Users of MQSeries for Windows NT Version 5.1 and later can perform the operations described in the following sections using the MQSeries Services snap-in. For more information, see Chapter 4, “Administration using the MQSeries Services snap-in” on page 37.

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

To display the status of the command server for a queue manager, called here `saturn.queue.manager`, the 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, which is the default.
- For an immediate stop, use the **endmqcsv** command with the -i flag.

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

Chapter 9. Administering remote MQSeries objects

This chapter describes how to administer MQSeries objects on a remote queue manager using MQSC commands as well as describing how you can use remote queue objects to control the destination of messages and reply messages.

For information about administration using the MQSeries Explorer, see Chapter 3, “Administration using the MQSeries Explorer” on page 29.

Channels, clusters, and remote queuing

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

- On the same machine
- On another machine in the same location or on the other side of the world
- Running on the same platform as the local queue manager
- Running on another platform supported by MQSeries

These messages may originate from:

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

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

- At least one channel
- A transmission queue
- A message channel agent (MCA)
- A channel listener
- A channel initiator

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

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

If the remote queue manager is expected to respond to messages sent by the local queue manager, a second channel needs to be set up to send responses back to the local queue manager.

Channels are defined using the MQSC DEFINE CHANNEL command. In this chapter, the examples relating to channels use the default channel attributes unless otherwise specified.

There is a message channel agent (MCA) at each end of a channel which controls the sending and receiving of messages. It is the job of the MCA to take messages from the transmission queue and put them on the communication link between the queue managers.

A transmission queue is a specialized local queue that temporarily holds messages before they are picked up by the MCA and sent to the remote queue manager. You specify the name of the transmission queue on a *remote queue definition*.

"Preparing channels and transmission queues for remote administration" on page 90 shows how to use these definitions to set up remote administration.

For more information about setting up distributed queuing in general, see the *MQSeries Intercommunication* book.

Remote administration using clusters

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

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

Once a cluster has been created the queue managers within it can communicate with each other *without the need for complicated channel or remote queue definitions*. Even the smallest cluster will reduce system administration overheads.

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

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

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

Remote administration from a local queue manager using MQSC commands

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

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

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 other MQSeries environments, including UNIX systems, AS/400, OS/390, OS/2, and Windows NT.

To implement remote administration, you must create specific 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 7 on page 90 shows the configuration of queue managers and channels that are required for remote administration using the **runmqsc** command. The object `source.queue.manager` is the *source* queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned. The object `target.queue.manager` is the name of the target queue manager, which processes the commands and generates any operator messages.

Note: If you are using MQSC with the -w option, `source.queue.manager` ***must*** be the default queue manager. For further information on creating a queue manager, see "crtmqm (Create queue manager)" on page 284.

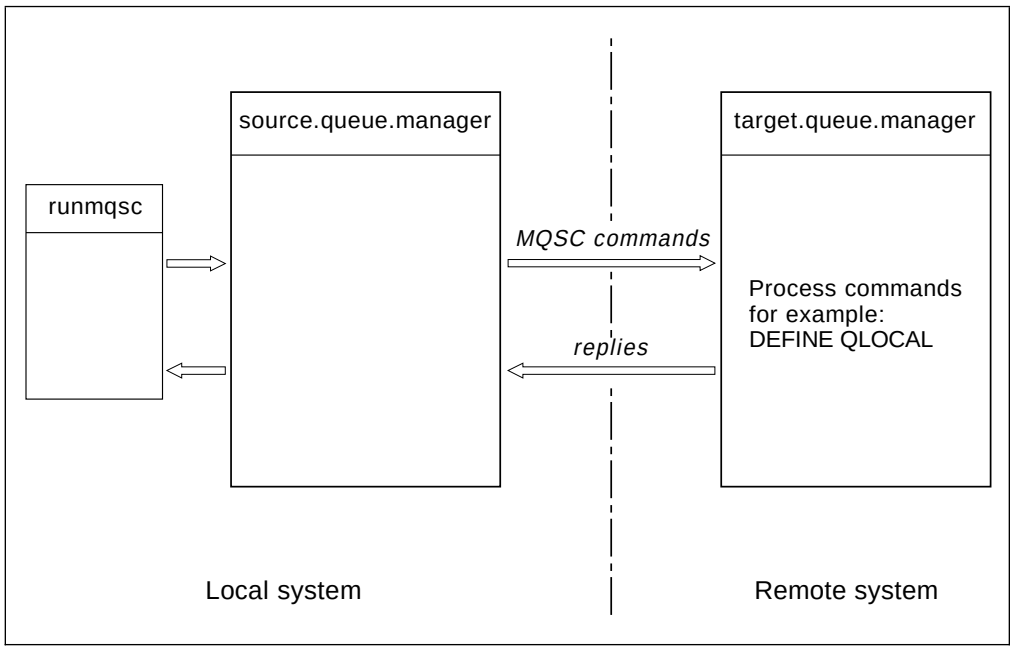


Figure 7. Remote administration using MQSC commands

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

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

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

On the target queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This queue is created by default when a queue manager is created.
- 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 target queue manager. 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. The XMITQ name must match the remote queue manager name in order for remote administration to work, unless you are using a queue manager alias.

Figure 8 on page 91 summarizes this configuration.

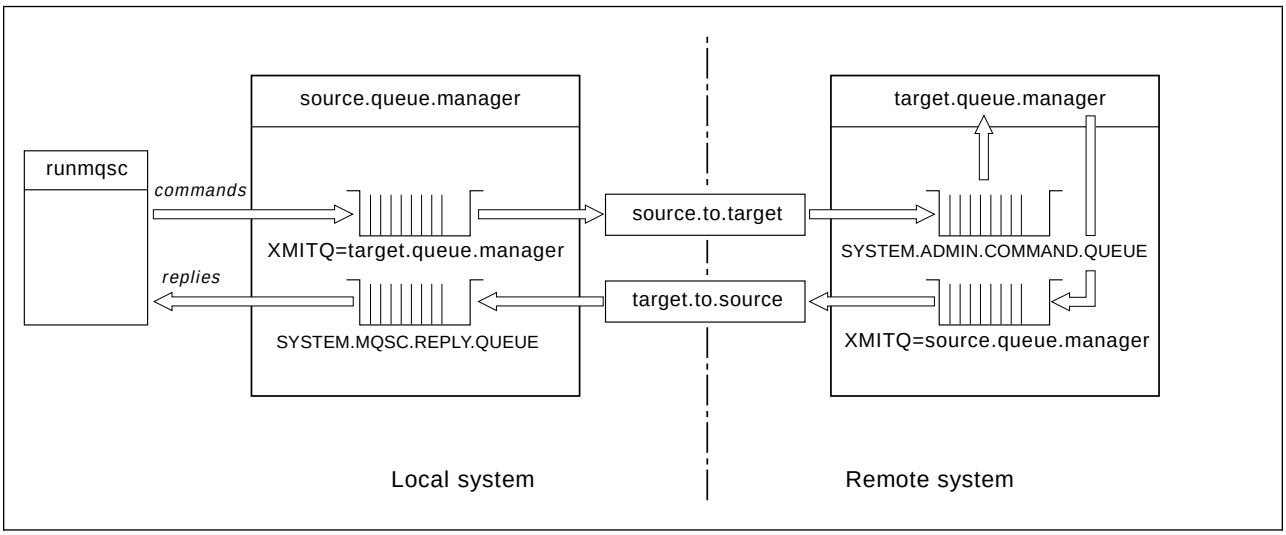


Figure 8. Setting up channels and queues for remote administration

See Chapter 1, “Concepts of intercommunication” in the *MQSeries Intercommunication* book 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 the commands shown in Figure 9 on page 92 on the target queue manager (`target.queue.manager`), to create the channels and the transmission queue there:

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

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

* Define the receiver channel on the target queue manager

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

* Define the transmission queue on the target queue manager

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

Figure 9. Commands to create channels and a transmission queue

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

Starting the channels

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

In MQSeries for UNIX systems, ensure that the `inetd` daemons have been configured for MQSeries and are running. Then start the channels as background processes:

- On the source queue manager, type:

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

- On the target queue manager, type:

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

In MQSeries for OS/2 Warp and Windows NT, start a listener as a background process at the receiver end of each channel.

- On the source queue manager, type:

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

- On the target queue.manager, type:

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

Then start the channels, again as background processes:

- On the source queue manager, type:

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

- On the target queue manager, type:

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

The **runmqlsr** and **runmqchl** commands are MQSeries control commands. They cannot be issued using **runmqsc**. Channels can however be started using **runmqsc** commands or scripts (start channel).

Automatic definition of channels

Automatic definition of channels applies only if the target queue manager is running on MQSeries Version 5.1 products. If an inbound attach request, or higher, is received and an appropriate receiver or server-connection definition cannot be found in the channel definition file (CDF), MQSeries creates a definition automatically and adds it to the CDF. Automatic definitions are based on two default definitions supplied with MQSeries: SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN.

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

For more information about the automatic creation of channel definitions, see "Auto-definition of channels" in the *MQSeries Intercommunication* book.

For information about the automatic definition of channels for clusters, see "Auto-definition of remote queues and channels" in the *MQSeries Queue Manager Clusters* book.

Issuing MQSC commands remotely

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

- On the target queue manager, type:

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

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

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

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

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

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

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

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

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

Working with queue managers on OS/390

You can issue MQSC commands to an OS/390 queue manager from an MQSeries Version 5.1 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 the source node:

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

You must define the receiver channel and the transmission queue at the source queue manager. The example assumes that TCP/IP is the transmission protocol being used.

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

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

Recommendations for remote queuing

When you are implementing remote queuing:

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

   You cannot use **runmqsc** to verify MQSC commands on another queue manager.
3. Check that the command file runs locally without error.
4. 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, use the following checklist to see if you have:

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

See also “Resolving problems with MQSC” on page 68.

Creating a local definition of a remote queue

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

Understanding how local definitions of remote queues work

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

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

Example

Purpose: An application 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, for example, `saturn.queue.manager`. The target queue is owned by another queue manager.

On the MQOPEN call, the application specifies these fields:

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

After this, the application issues an MQPUT call to put a message 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')`
: Additional text that describes the use of the queue.

`RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)`
: Specifies the name of the target queue on the remote queue manager. This is the real target queue for messages 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)
: Specifies the name of the remote queue manager that owns the target queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE.

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

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

An alternative way of putting messages on a remote queue

Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, which includes the remote queue manager name, as part of the MQOPEN call. 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.

DISPLAY QUEUE (CYAN.REMOTE.QUEUE)


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

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


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

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

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

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 target 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” on page 95.

Using remote queue definitions as aliases

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

- Queue manager aliases
- Reply-to queue aliases

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

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

Queue manager aliases

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

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

For more information about queue manager aliases, see “Queue manager alias definitions” in the *MQSeries Intercommunication* book.

Reply-to queue aliases

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

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

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

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

For more information about request messages, reply messages, and reply-to queues, see Chapter 3, “MQSeries messages” in the *MQSeries Application Programming Guide*.

For more information about reply-to queue aliases, see “Reply-to queue alias definitions” in the *MQSeries Intercommunication* book.

Data conversion

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

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

For EBCDIC new line (NL) character conversions to ASCII, see "The AllQueueManagers stanza" on page 130.

Supported conversions are defined in Appendix F, "Code page conversion tables" in the *MQSeries Application Programming Reference* manual.

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

The queue manager cannot automatically convert messages in built-in formats if their CCSIDs represent different national-language groups. For example, conversion between CCSID 850 and CCSID 1025 (which is an EBCDIC coded character set for languages using Cyrillic script) is not supported because many of the characters in one coded character set cannot be represented in the other. If you have a network of queue managers working in different national languages, and data conversion among some of the coded character sets is not supported, you can enable a default conversion. Default data conversion is described in "Default data conversion" on page 101.

File ccsid.tbl

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

- In MQSeries for Windows NT it records all the supported code sets. In MQSeries for OS/2 Warp and UNIX systems the supported code sets are held internally by the operating system.
- It specifies any additional code sets. To specify additional code sets, you need to edit ccsid.tbl (guidance on how to do this is provided in the file).
- It specifies any default data conversion.

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

In UNIX environments, a sample ccsid.tbl file is provided as **mqmtop**/samp/ccsid.tbl.

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

In MQSeries for OS/2 Warp and MQSeries for Windows NT, ccsid.tbl is located on the boot drive in directory \MQM\CONV\TABLE.

Default data conversion

To implement default data conversion, you edit ccsid.tbl to specify a default EBCDIC CCSID and a default ASCII CCSID, and also to specify the defaulting CCSIDs. Instructions for doing this are included in the file.

If you update ccsid.tbl to implement default data conversion, the queue manager must be restarted before the change can take effect.

The default data-conversion process is as follows:

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

Note: You should try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. If you use only that set of characters that is valid for MQSeries object names (as defined in “Names of MQSeries objects” on page 279) you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

Conversion of messages in user-defined formats

Messages in user-defined formats cannot be converted from one coded character set to another by the queue manager. If data in a user-defined format requires conversion, you must supply a data-conversion exit for each such format. The use of default CCSIDs for converting character data in user-defined formats is not recommended, although it is possible. For more information about converting data in user-defined formats and about writing data conversion exits, see Chapter 11, “Writing data-conversion exits” in the *MQSeries Application Programming Guide*.

Chapter 10. Protecting MQSeries objects

This information does not apply to MQSeries for OS/2 Warp

This chapter describes how to prevent unauthorized access to MQSeries objects in these environments:

- MQSeries for AIX
- MQSeries for HP-UX
- MQSeries for Sun Solaris
- MQSeries for Windows NT

Detailed information about installable services is given in the Chapter 11, “Installable services and components” in the *MQSeries Programmable System Management* manual.

This chapter contains these sections:

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, clusters, 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.

Before you begin (UNIX systems)

In MQSeries for UNIX systems, UNIX restrictions mean that all user IDs must be defined in lowercase.

All queue manager processes run with these IDs:

```
User ID      mqm
Group        mqm
```

A user ID with the name mqm whose primary group is mqm is automatically created during installation. You can create the user ID and group yourself, but you must do this before you install MQSeries.

For an explanation of how to create the ID and group yourself, see one of the following:

- "Chapter 3. Installing the MQSeries for AIX Server" in the *MQSeries for AIX V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for HP-UX Server" in the *MQSeries for HP-UX V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for Sun Solaris Server" in the *MQSeries for Sun Solaris V5.1 Quick Beginnings* book

User IDs in user group mqm (UNIX systems)

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

- Use the **runmqsc** command to run MQSC commands
- Administer authorities using the **setmqaut** command
- Create a queue manager using the **crtmqm** command

If you are sending channel commands to remote queue managers, you must make sure that your user ID is a member of group mqm on the target system. For a list of PCF and MQSC channel commands, see "Channel command security" on page 115.

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. However, the special MQI call, MQCONNX, **does** require mqm group membership if the option MQCNO_FASTPATH_BINDING is used.

Before you begin (Windows NT)

If the local mqm group does not already exist on the local computer, it is created automatically when MQSeries for Windows NT is installed. In addition, a Domain mqm group may be created on the domain controller. This global group allows control of mqm user access. All privileged user IDs active within this domain should be added to the Domain mqm group.

User IDs for administration

If your user ID belongs to the local mqm or Administrators group, you can administer any queue manager on that system. The system-defined user ID 'SYSTEM' can also administer any queue manager.

The name of the local mqm group to be used for privileged MQSeries administration is fixed, and it can contain (directly, or indirectly by the inclusion of global groups) users who require MQSeries authority to any queue manager on the workstation or server.

In order to run all the MQSeries for Windows NT control commands, your user ID must belong to the local mqm or Administrators group. In particular, you need this authority to:

- Use the **runmqsc** command to run MQSC commands
- Administer authorities on MQSeries for Windows NT using the **setmqaut** command
- Create a queue manager using **crtmqm**

If you are sending channel commands to queue managers on a remote Windows NT system, you must ensure that your user ID is a member of the mqm or Administrators group on the target system. For a list of PCF and MQSC channel commands, see "Channel command security" on page 115.

Some control commands, for example, **crtmqm**, manipulate authorities on MQSeries objects using the Object Authority Manager (OAM). As described in "Understanding the Object Authority Manager" on page 107, the OAM uses a predefined search order to determine the authority rights for a given user ID. Consequently the authorities granted to your user ID may differ from those determined by the OAM. For example, if you issue **crtmqm** from a user ID authenticated by a domain controller that has membership of the local mqm group through a global group, the command fails if the system has a local user of the same name who is not in the local mqm group.

Your user ID does not have to belong to group mqm in order to issue:

- PCF commands—including Escape PCFs—from an administration program.
- MQI calls from an application program. However, the special MQI call, MQCONNX, **does** require mqm group membership.

When you use a Domain user ID defined on a remote machine, you must be a member of the local mqm or Administrators group to:

1. Issue commands (such as create queue manager), and
2. Grant MQSeries authorities.

Name lengths for user IDs and groups

For MQSeries authorizations, names of user IDs, groups, and domains are limited to:

- 20 characters for user IDs
- 64 characters for group names
- 15 characters for domain names

Qualifying a user ID with a domain name

You can optionally qualify a user ID on both the **setmqaut** and **dspmqaut** commands with a domain name using the following syntax:

```
user@domain
```

For example,

```
setmqaut -m qmgrname -t qmgr -p userID@domain +all
```

where `user` is the user ID and `domain` is the domain name.

Note: Group names always refer to local groups, therefore domain qualification is not necessary.

Using the @ symbol in user ID names

The at sign (@ symbol) is used as a delimiter. However, some user IDs may contain the @ symbol as part of the user ID name. If you do this, use two @ symbols together (@@) to signify that the symbol is to be used as part of the user ID string and not as a delimiter between the user ID and the domain name. So, for example, the user ID `a@b` becomes `a@@b`.

Using spaces in names for user IDs and groups

Spaces in user IDs and group names are allowed when they are specified as parameters on the **setmqaut** and **dspmqaut** commands, as long as they are enclosed in double quotation marks (" ").

Authorizing user IDs on different domains

You can give different levels of authorization to user IDs that are not unique residing on different domains. For example, user ID Fred on domain A can be given different authorizations to user ID Fred on domain B.

See "dspmqaut (Display authority)" on page 293, and "setmqaut (Set/reset authority)" on page 327 for command descriptions.

Restricted-access Windows NT objects

When MQSeries creates restricted-access Windows NT objects, full control permission is given to the following entities:

- The local mqm group on the local computer
- The local Administrators group on the local computer
- The SYSTEM user ID

Security policies

A security policy can be specified for each queue manager by setting the SecurityPolicy attribute of the `Service` stanza in the Windows NT Registry. See "The Service stanza" on page 136 for a description of the attribute.

A security policy dictates how the OAM behaves when it receives authority requests which do not contain Windows NT security identifier (NT SID) information[3].

When using the default security policy, it is permissible for the OAM to receive authority requests that do not contain SID information. In such situations, the OAM attempts to resolve the user ID into a Windows NT SID by searching:

- The local security database
- The security database of the primary domain
- The security database of trusted domains

If the security policy is set to the value NTSIDsRequired, then both the user ID and NT SID information must be passed to the OAM.

In cases where both a user ID and NT SID information are passed to the OAM, a check is made to ensure that the two are consistent. The supplied user ID is compared with the user ID (or the first 12 characters if the user ID is longer than 12 characters) associated with the NT SID. If the two are unequal, then authorization fails. This consistency check is performed regardless of the security policy setting.

Understanding the Object Authority Manager

By default, access to queue-manager resources is controlled through an authorization service installable component formally called the Object Authority Manager (OAM) for MQSeries. It is supplied with MQSeries, 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 MQSeries.

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 Chapter 11, “Installable services and components” in the *MQSeries Programmable System Management* manual.

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

[3] The Windows NT security identifier (NT SID) supplements the 12-character user ID. It contains information that identifies the full user account details on the Windows NT security account manager (SAM) database where the user is defined. When a message is created on MQSeries for Windows NT, MQSeries stores the SID in the message descriptor. When MQSeries for Windows NT performs authorization checks, it uses the SID to query the full information from the SAM database. The SAM database in which the user is defined must be accessible for this query to succeed.

How the OAM works

The OAM works by exploiting the security features of the underlying operating system. In particular, the OAM uses operating system 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 are always user IDs.

Group sets and the primary group

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

Group sets and the primary group—MQSeries for UNIX systems: One of the groups in the group set is the *primary group*.

Group sets and the primary group—MQSeries for Windows NT: For MQSeries for Windows NT, the role of the primary group is fulfilled by the user ID. The Windows NT primary group associated with a user ID is given no special treatment by MQSeries; it is handled in the same way as any other group.

| The OAM searches for the specified user in the following order:

| 1. The local security database
| 2. The security database of the primary domain
| 3. The security database of trusted domains

The first user ID encountered is used when checking for group membership.

Note that each of these user IDs may have different group memberships on a particular computer.

When a principal belongs to more than one group

The authorizations that a principal has are derived from the union of the authorizations of its group set. Whenever a principal requests access to a resource, the OAM computes this union and uses the resultant authorization to check the principal's access to the resource. You can use the control command **setmqaut** to set the authorizations for a specific principal. However, for MQSeries for UNIX systems, this also gives the same authorizations to the principal's primary group.

The group set associated with a principal is cached when the group authorizations
| are computed by the OAM. Any changes made to a principal's group memberships after the group set has been cached are not recognized until the queue manager is restarted.

Default user group (UNIX systems only)

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 that the user ID making the request has the authorization for the operation requested.

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

- Permission to use PCF commands.

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

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 their user ID to the appropriate group.

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.

Notes:

1. In MQSeries for UNIX systems, if a principal in a PRIMARY group is added to the MQM group, then all members of the PRIMARY group inherit the authority of the member added, unless you use SETMQAUT to change the authority of the existing members. It is 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.
2. MQSeries for Windows NT treats the local Administrators group and the local mqm group in a special manner. Members of these groups are always granted full access rights which cannot be removed. Membership of these local groups may be established by the user being a member of a domain global group which is included in the local group.

Disabling the object authority manager

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

In MQSeries for UNIX systems, you set MQSNOAUT as follows:

```
export MQSNOAUT=yes
```

For MQSeries for Windows NT, you set MQSNOAUT as follows:

```
SET MQSNOAUT=yes
```

However, if you do this you cannot, in general, restart the OAM later. A better approach is to have the OAM enabled and ensure that all users and applications have access through an appropriate group or 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 Object Authority Manager commands

The OAM provides a command interface for granting and revoking authority. Before you can use these commands, you must be suitably authorized:

- In MQSeries for UNIX systems, your user ID must belong to the group mqm, which you define when you install MQSeries.
- In MQSeries for Windows NT, your user ID must belong to either the local mqm group or the local Administrators group.

If your user ID is a member of mqm, or, for MQSeries for Windows NT, of either mqm or the local Administrators group, you have a ‘super user’ authority to the queue manager, which means that you are authorized to issue any MQI request or command from your user ID.

Using the OAM set or reset authority control command, setmqaut

The OAM provides two control commands that allow you 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. Therefore, any changes made to the authority of an object using **setmqaut** do not take effect until you reset the object.

The authority commands **setmqaut** and **dspmqaut** apply to the specified queue manager; if you do not specify the name of a queue manager, the default queue manager is assumed. On these commands, you must also identify 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 of identifying the resources to which the change in authorization applies. Each authorization in the list is specified as a lowercase keyword, prefixed with a plus sign (**+**) or a minus sign (–). Use a plus sign to add the specified authorization, and a minus sign to remove the authorization. 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:

- `saturn.queue.manager` is the queue manager name.
- `queue` is the object type.
- `RED.LOCAL.QUEUE` is the object name.
- `groupa` is the ID of the group whose authorizations are to change.
- `+browse -get +put` is the authorization list for the specified queue. There must be no spaces between the '+' or '–' signs and the keyword.
  - `+browse` adds authorization to browse messages on the queue (to issue MQGET with the browse option).
  - `-get` removes authorization to get (MQGET) messages from the queue.
  - `+put` adds authorization to put (MQPUT) messages on the queue.

  In summary, applications started with user IDs that belong to user group `groupa` have at least these authorizations.

You can specify one or more principals and, at the same time, one or more groups. For example, the following command revokes put authority on the queue `MyQueue` from the principal `fvuser` and from groups `groupa` and `groupb`.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser -g groupa -g groupb -put
```

Note: For MQSeries for UNIX systems, 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)" on page 327.

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.

See "setmqaut (Set/reset authority)" on page 327 for detailed command information.

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)" on page 327.

Using the OAM display authority control command (dspmqaut)

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 be displayed for only one group or principal at a time. See "dspmqaut (Display authority)" on page 293 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,

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

User IDs (MQSeries for UNIX systems only)

The special user ID mqm that you create is intended for use by the product only. It should never be available to nonprivileged users.

If an MQ process is associated with a login session, then the authorization routines check the real (logged-in) user ID.

If an MQ process is not associated with a login session (for example, if the process is invoked from a daemon such as inetd), the effective user ID is used for authorization. In a CICS environment, the CICS user ID associated with the transaction is used.

All objects are owned by user ID mqm.

Queue manager directories

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

Queues

The authority to a dynamic queue is based on, but is not necessarily the same as, that of the model queue from which it is derived. See note 1 on page 119 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 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, 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 “Message context” in the *MQSeries Application Programming Guide*.

For descriptions of the message descriptor fields relating to context, see “MQMD - Message descriptor” on page 109 in 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 Table 4 on page 118.

Channel exits Channel exits can be used for added security.

For more information about remote security, see Chapter 6, “Channel attributes” and Chapter 35, “Channel-exit programs” in the *MQSeries Intercommunication* book.

Channel command security

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

PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote MQSeries system. The user ID, as specified in the message descriptor of the PCF message, must belong to group mqm (or the Administrator’s group in the MQSeries for Windows NT) on the target system. These commands are:

- *ChangeChannel*
- *CopyChannel*
- *CreateChannel*
- *DeleteChannel*
- *PingChannel*
- *ResetChannel*
- *StartChannel*
- *StartChannelInitiator*
- *StartChannelListener*
- *StopChannel*
- *ResolveChannel*

See “Authority checking for PCF commands” in the *MQSeries Programmable System Management* manual for the PCF security requirements.

MQSC channel commands

You can issue MQSC channel commands to a remote MQSeries system either by sending the command directly in a PCF escape message or by issuing the command using **runmqsc** in indirect mode. The user ID as specified in the message descriptor of the associated PCF message must belong to group `mqm` (or the Administrator's group in MQSeries for Windows NT) 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
- START LISTENER
- 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 (or the Administrator's group in MQSeries for Windows NT). These commands are:

- **runmqchi** (Run channel initiator)
- **runmqchl** (Run channel)
- **runmqlsr** (Run listener)

Understanding the authorization specification tables

The authorization specification tables starting on page 118 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:

- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following:

Action to be performed MQI option, MQSC command, or PCF command.

Access control object Queue, process, 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 “Authorization file contents — MQSeries for UNIX systems” on page 124 for more information.

MQI authorizations

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

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

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

Table 4 on page 118 summarizes the authorizations needed for each call.

Table 4. Security authorization needed for MQI calls

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

Notes for Table 4:

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

    Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all of the following that are relevant to the object type:
   - MQZAO_CONNECT
   - MQZAO_INQUIRE
   - MQZAO_SET
   - MQZAO_BROWSE
   - MQZAO_INPUT
   - MQZAO_OUTPUT
   - MQZAO_PASS_IDENTITY_CONTEXT
   - MQZAO_PASS_ALL_CONTEXT
   - MQZAO_SET_IDENTITY_CONTEXT
   - MQZAO_SET_ALL_CONTEXT
   - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note 14) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. 'No check' means that no authorization checking is carried out.
4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Administration authorizations

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

Authorizations for MQSC commands in escape PCFs

Table 5 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

Table 5. MQSC commands and security authorization needed

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

Notes for Table 5:

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, namelist, or queue manager object is checked, depending on the type of object.
3. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the **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

Table 6 summarizes the authorizations needed for each PCF command.

Table 6. PCF commands and security authorization needed

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

Notes for Table 6:

1. The user identifier under which the program submitting the command is running must also have authority to connect to its local queue manager, and to open the command administration queue for output.

2. Either the queue, process, namelist, or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the **setmqaut** command.
5. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
6. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

General notes:

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

   MQZAO_CREATE is not included because it is not specific to a particular object or object type.
3. 'No check' means that no authorization checking is carried out.
4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot use a Clear Queue command on a process object.

Authorization files

Attention!

The information in this section is given for problem determination purposes. Under normal circumstances, use authorization commands to view and change authorization information.

MQSeries uses a specific file structure to implement security. You should 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.

Types of authorization

There are three types of authorization:

- Authorizations applying to single objects, 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 users.

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 subdirectory, the path of which is defined by:

- The queue manager name
- The type of authorization
- 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

In MQSeries for UNIX systems, the default authorization directories for a queue manager called `saturn` are:

Table 7. Authorization directories for MQSeries for UNIX systems

Authorization directory	Description
`/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/namelist`	Authorization files for the namelists

In MQSeries for Windows NT, the default authorization directories for a queue manager called `saturn` are:

Table 8. Authorization directories for MQSeries for Windows NT

Authorization directory	Description
`\mqm\qmgrs\saturn\auth\queues`	Authorization files for queues.
`\mqm\qmgrs\saturn\auth\procdef`	Authorization files for process definitions.
`\mqm\qmgrs\saturn\auth\qmanager`	Authorization files for the queue manager.
`\mqm\qmgrs\saturn\auth\namelist`	Authorization files for the namelists.

In the auth directory, @class files hold the authorizations related to the entire class.

There is a difference between @class (the authorization file that specifies authorization for a particular class) and @aclass (the authorization file that specifies authorizations to all classes).

Paths for object authorization files

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
```

If the name and path of SYSTEM.DEFAULT.LOCAL.QUEUE is:

```
\mqm\qmgrs\saturn\queues\SYSTEM!DEFAULT!LOCAL!QUEUE
```

the name and path of the corresponding authorization file is:

```
\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 “Understanding MQSeries file names” on page 27 for details.

Authorization file contents — MQSeries for UNIX systems

The authorizations of a particular group are defined by a set of stanzas in the authorization file. The authorizations apply to the object associated with this file. For example:

```
groupB:
   Authority=0x0040007
```

This stanza defines the authority of the group `groupB`.

Authorization file contents — MQSeries for Windows NT

The authorizations of a particular user ID or group are defined by a set of attributes in the authorization file. The authorizations apply to the object associated with this file. For example:

```
user@domain1:
 Authority = 0x0040007
 Sid = S-1-5-21-1023809979-1377598139-60295696-1024
```

This stanza defines the authority of the user ID `user`.

Authority stanza

The authority specification is the union of the individual bit patterns based on the assignments shown in Figure 10

```
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
alladm               MQZAO_ALL_ADMIN                      0x009E0000
none                 MQZAO_NONE                           0x00000000
all                  MQZAO_ALL                            0x009E07FF
```

Figure 10. Authority specification

These definitions are made in the header file cmqzc.h.

In the following example, `groupB` and `user` have 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)

They also have DISPLAY authority for the object associated with this authorization file.

Class authorization files — MQSeries for UNIX systems and MQSeries for Windows NT

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:

- The entry MQZAO_ALTERNATE_USER_AUTHORITY gives authorization to assume the identity of another user when interacting with MQSeries objects.
- The entry MQZAO_SET_ALL_CONTEXT gives authorization to set the context of a message when issuing MQPUT.
- The entry MQZAO_SET_IDENTITY_CONTEXT gives authorization to set the identity context of a message when issuing MQPUT.

Managing authorization files

Here are some items that you need consider when managing your authorization files:

1. You must ensure that the authorization files are secure and not write-accessible 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 or command files containing the commands used
3. You can copy and edit authorization files. However, you should not normally have to create or repair them manually. Should an emergency occur, you can use the information given here to recover lost or damaged authorization files.

Authorizations to authorization files

In MQSeries for UNIX systems, 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, your user ID must either be mqm, or it must belong to the mqm group.

For MQSeries for Windows NT, authorization files must be readable by any principal. However, only the mqm or Administrator's group should be allowed to update these files.

To alter authorizations using the **setmqaut** command supplied with MQSeries for Windows NT, your Windows NT user ID must belong to the local mqm group or the local Administrators group.

Chapter 11. Configuring MQSeries

This chapter explains how to change the behavior of an individual queue manager, or of a node, to suit your installation's needs.

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

How you change this configuration information, and where MQSeries stores your changes, is platform-specific:

- MQSeries for Windows NT uses the MQSeries Services snap-in to make changes to attribute information within the **Windows NT Registry**.
- Users on all other platforms change attribute values by editing the **MQSeries configuration files**.

This chapter:

- Describes the platform-specific methods for reconfiguring MQSeries in:
  - "MQSeries configuration files" (for MQSeries for UNIX systems and MQSeries for OS/2 Warp)
  - "Editing configuration information" on page 26
- Describes the attributes you can use to modify MQSeries configuration information in "Attributes for changing MQSeries configuration information" on page 130.
- Describes the attributes you can use to modify queue manager configuration information in "Changing queue manager configuration information" on page 136.
- Provides examples of mqs.ini and qm.ini files for MQSeries for UNIX systems and MQSeries for OS/2 Warp in "Example mqs.ini and qm.ini files for MQSeries for OS/2 Warp" on page 149 and "Example mqs.ini and qm.ini files for MQSeries for UNIX systems" on page 154.

MQSeries configuration files

Users of platforms other than MQSeries for Windows NT modify MQSeries configuration attributes within:

- An MQSeries configuration file (**mqs.ini**) to effect changes for MQSeries on the node as a whole. There is one mqs.ini file per node.
- A queue manager configuration file (**qm.ini**) to effect changes for specific queue managers. There is one qm.ini file for each queue manager on the node.

A configuration file (which can be referred to as a ***stanza*** file) contains one or more stanzas, which are simply groups of lines in the .ini file that together have a common function or define part of a system, for example, log functions, channel functions, and installable services.

Any changes you make to a configuration file will not take effect until the next time the queue manager is started.

Editing configuration files

Before attempting to edit a configuration file, back it up so that you have a copy you can revert to if the need arises!

You can edit configuration files either:

- Automatically, using commands that change the configuration of queue managers on the node
- Manually, using a standard text editor

You can edit the default values in the MQSeries configuration files after installation.

If you set an incorrect value on a configuration file attribute, the value is ignored and an operator message is issued to indicate the problem. (The effect is the same as missing out the attribute entirely.)

When you create a new queue manager, you should:

- Back up the MQSeries configuration file
- Back up the new queue manager configuration file

When do you need to edit a configuration file?

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

- You lose a configuration file; recover from backup if possible.
- You need to move one or more queue managers to a new directory.
- You need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- You are advised to do so by your IBM Support Center.

Configuration file priorities

The attribute values of a configuration file are set according to the following priorities:

- Parameters entered on the command line take precedence over values defined in the configuration files
- Values defined in the qm.ini files take precedence over values defined in the mqs.ini file.

The MQSeries configuration file, mqs.ini

The MQSeries configuration file, mqs.ini, contains information relevant to all the queue managers on the node. It is created automatically during installation. In particular, the mqs.ini file is used to locate the data associated with each queue manager.

When installing MQSeries, you can specify two target directories: one for programs and one for data. The mqs.ini file is stored in the data directory, **mqm**.

These directories can be on different drives, although this is not mandatory. However, to improve performance, it is best that the directories reside on different drives.

The default mqm directory for:

- MQSeries for UNIX systems can be found at /var/mqm
- MQSeries for OS/2 Warp is specified on the MQSWORKPATH environment variable which is set at install time. The default is

  ```
  <bootdrive>:\MQM
  ```

  For many OS/2 machines this is usually C:\MQM

The mqs.ini file contains:

- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each of them.

Queue manager configuration files, qm.ini

A queue manager configuration file, qm.ini, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. The qm.ini file is automatically created when the queue manager with which it is associated is created.

A qm.ini file is held in the root of the directory tree occupied by the queue manager.

For example, in an MQSeries for UNIX systems system, the path and the name for a configuration file for a queue manager called QMNAME is:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

For MQSeries for OS/2 Warp, the path and name for configuration file for a queue manager called QMNAME is:

```
C:\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” on page 27.

Attributes for changing MQSeries configuration information

The following groups of attributes appear in mqs.ini and have equivalents in the Windows NT Registry:

The AllQueueManagers stanza

The `AllQueueManagers` stanza can specify:

- The path to the qmgrs directory where the files associated with a queue manager are stored
- The path to the executable and DLL libraries
- The method for converting EBCDIC-format data to ASCII format

DefaultPrefix=*directory_name*

This attribute specifies the path to the qmgrs directory, below which the queue manager data is kept.

If you change the default prefix for the queue manager, you must replicate the directory structure that was created at installation time (see Figure 64 on page 348).

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

As an alternative to changing the default prefix, you can use the environment variable MQSPREFIX to override the `DefaultPrefix` for the **crtmqm** command.

DefaultFilePrefix=*path* **(OS/2 only)**

This attribute specifies the path where the DLLs can be found.

ConvEBCDICNewline=NL_TO_LF|TABLE|ISO

EBCDIC code pages contain a new line (NL) character that is not supported by ASCII code pages; although some ISO variants of ASCII do contain an equivalent.

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

NL_TO_LF

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

NL_TO_LF is the default.

TABLE

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

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

ISO
Specify ISO if you want:

- ISO CCSIDs to be converted using the TABLE method
- All other CCSIDs to be converted using the NL_TO_CF method.

Possible ISO CCSIDs are shown in Table 9 on page 131.

Table 9. List of possible ISO CCSIDs

CCSID	Code Set
819	ISO8859-1
912	ISO8859-2
915	ISO8859-5
1089	ISO8859-6
813	ISO8859-7
916	ISO8859-8
920	ISO8859-9
1051	roman8

If the ASCII CCSID is not an ISO subset, ConvEBCDICNewline defaults to NL_TO_LF.

For more information about data conversion, see “Application data conversion” in the *MQSeries Application Programming Guide*.

The ClientExitPath stanza

The `ClientExitPath` stanza specifies the default path for location of the channel exit on the client. This stanza applies to MQSeries clients on AIX, HP-UX, OS/2, Sun Solaris, and the Windows 3.1 client. The client server information for MQSeries for Windows NT is now in the Windows NT Registry.

ExitsDefaultPath=*defaultprefix*
The ExitsDefaultPath attribute specifies the default prefix for the platform. For example, for OS/2 this could be C:\mqm\exits

The DefaultQueueManager stanza

The `DefaultQueueManager` stanza specifies the default queue manager for the node.

Name=*default_queue_manager*
The default queue manager processes any commands for which a queue manager name is not explicitly specified. The `DefaultQueueManager` attribute is automatically updated if you create a new default queue manager. If you inadvertently create a new default queue manager and then want to revert to the original, you must alter the `DefaultQueueManager` attribute manually.

The ExitProperties stanza

The `ExitProperties` stanza specifies configuration options used by queue manager exit programs.

CLWLmode=SAFE|FAST
The cluster workload exit, CLWL, allows you to specify which cluster queue in the cluster is to be opened in response to an MQAPI call (MQOPEN or MQPUT and so on). The CLWL exit runs either in FAST mode or SAFE mode depending on the value you specify on the CLWLMode attribute. If the CLWLMode attribute is not specified, the cluster workload exit runs in SAFE mode.

SAFE
The SAFE option specifies that the CLWL exit is to run in a separate process to the queue manager. This is the default.

If a problem arises with the user-written CLWL exit when running in SAFE mode, the following happens:

- The CLWL server process (amqzlwa0) fails
- The queue manager restarts the CLWL server process
- The error is reported to you in the error log. If an MQAPI call is in progress, you receive notification in the form of a bad return code.

The integrity of the queue manager is preserved.

Note: There is an overhead associated with running the CLWL exit in a separate process, which can affect performance.

FAST
Specify FAST if you want the cluster exit to run inline in the queue manager process.

Specifying this option improves performance by avoiding the overheads associated with running in SAFE mode, but does so at the expense of queue manager integrity. Therefore, you should only run the CLWL exit in FAST mode if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance overheads.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager will fail and you run the risk of the integrity of the queue manager being compromised.

The LogDefaults stanza

The `LogDefaults` stanza specifies the default log attributes for the node. The log attributes are used as default values when you create a queue manager, but can be overridden if you specify the log attributes on the **crtmqm** command. See "crtmqm (Create queue manager)" on page 284 for details of this command.

Once a queue manager has been created, the log attributes for that queue manager are read from its log stanza in the qm.ini file.

The *DefaultPrefix* attribute (in the `AllQueueManagers` stanza) and the *LogPath* attribute in the `LogDefaults` stanza allow for the queue manager and its log to be

on different physical drives. This is the recommended method, although, by default, they are on the same drive.

For information about calculating log sizes, see “Calculating the size of the log” on page 219.

Note: The limits given in the following parameter list are limits set by MQSeries. Operating system limits may reduce the maximum possible log size.

LogPrimaryFiles=3|*2-62*
Primary log files are the log files allocated during creation for future use.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogSecondaryFiles=2|*1-61*
Secondary log files are the log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

For MQSeries for UNIX systems, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages 64 and the maximum is 16 384.

For MQSeries for OS/2 Warp and MQSeries for Windows NT, the default number of log file pages is 256, giving a log file size of 1 MB. The minimum number of log file pages is 32 and the maximum is 4095.

LogType=CIRCULAR|LINEAR
The `LogType` attribute is used to define the type to be used. The default is CIRCULAR.

CIRCULAR
Set this value if you want to start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Circular logging” on page 215 for a fuller explanation of circular logging.

LINEAR
Set this value if you want both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See “Linear logging” on page 215 for a fuller explanation of linear logging.

If you want to change the default, you can either edit the `LogType` attribute, or specify linear logging using the **crtmqm** command. You cannot change the logging method after a queue manager has been created.

LogBufferPages=17|*4-32*

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 minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

The default number of buffer pages is 17, equating to 68 KB.

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.

LogDefaultPath=*directory_name*

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 in MQSeries for UNIX systems
- <DefaultPrefix>\LOG for MQSeries for OS/2 Warp and MQSeries for Windows NT where <DefaultPrefix> is the value specified on the DefaultPrefix attribute in the `AllQueueManagers` stanza of the mqs.ini file. This value is set at install time, and by default is

  `<bootdrive>\:MQM`

  For many machines, this is

  `C:\MQM\LOG`

Alternatively, you can specify the name of a directory on the **crtmqm** command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the Log File Path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the **crtmqm** command, the value of the `LogDefaultPath` attribute in the mqs.ini file is used.

The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager has been created, a `LogPath` value is created in the log attributes in the qm.ini 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.

The QueueManager stanza

There is one `QueueManager` stanza for every queue manager. These attributes specify the queue manager name, and the name of the directory containing the files associated with that queue manager. The name of the directory is based on the queue manager name, but is transformed if the queue manager name is not a valid file name.

See "Understanding MQSeries file names" on page 27 for more information about name transformation.

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

Prefix=*prefix*
This attribute specifies where the queue manager files are stored. By default, this is the same as the value specified on the DefaultPrefix attribute of the `AllQueueManager` stanza in the mqs.ini file.

Directory=*name*
This attribute specifies the name of the subdirectory under the `<prefix>\QMGRS` directory where the queue manager files are stored. This name is based on the queue manager name but can be transformed if there is a duplicate name, or if the queue manager name is not a valid file name.

Changing queue manager configuration information

The following groups of attributes can appear in a qm.ini file particular to a given queue manager, or used to override values set in mqs.ini.

The Service stanza

The `Service` stanza specifies the name of an installable service, and the number of entry points to that service. There must be one `Service` stanza for every service used.

For each component within a service, there must be a `ServiceComponent` stanza, which identifies the name and path of the module containing the code for that component. See "The ServiceComponent stanza" for more information.

Name=AuthorizationService|NameService|UserIDService
Specifies the name of the required service.

AuthorizationService
For MQSeries, the Authorization Service component is known as the Object Authority Manager, or OAM.

- In MQSeries for UNIX systems, the `AuthorizationService` stanza and its associated `ServiceComponent` stanza are added automatically when the queue manager is created, but can be overridden through the use of `mqsnoaut`. Any other `ServiceComponent` stanzas must be added manually.
- In MQSeries for Windows NT systems, each queue manager has its own key in the Windows NT Registry. The equivalents for the `Service` and `ServiceComponent` stanzas for the default authorization component are added to the Windows NT Registry automatically, but can be overridden through the use of the `mqsnoaut` environment variable. Any other `ServiceComponent` stanzas must be added manually.
- For MQSeries for OS/2 Warp, no authorization service component is supplied with the product. Therefore, by default, no `Service` and `ServiceComponent` stanzas are added to the qm.ini file. Facilities exist for you to write your own authorization service component which you then add as a AuthorizationService stanza to the qm.ini file manually to enable that service. To disable the service, delete the relevant group of attributes.

NameService
The `NameService` stanza must be added to the qm.ini file manually to enable the supplied name service.

UserIDService (MQSeries for OS/2 Warp only)
The `UserIDService` stanza must be added to the qm.ini file manually to enable the service.

EntryPoints=*number-of-entries*
Specifies the number of entry points defined for the service. This includes the initialization and termination entry points.

SecurityPolicy=Default|NTSIDsRequired (MQSeries for Windows NT only)
The SecurityPolicy attribute is applicable only if the service specified on the `Service` stanza is the authorization service, that is, the default OAM. The SecurityPolicy attribute allows you to specify the security policy for each queue manager. The possible values are:

Default
Specify `Default` if you want the default security policy to take effect. If a Windows NT security identifier (NT SID) is not passed to the OAM for a particular user ID, then an attempt is made to obtain the appropriate SID by searching the relevant security databases.

NTSIDsRequired
Requires that an NT SID is passed to the OAM when performing security checks.

See "Security policies" on page 106 for more information.

For more information about installable services and components, see Chapter 11, "Installable services and components" in the *MQSeries Programmable System Management* book.

For more information about security services in general, see Chapter 10, "Protecting MQSeries objects" on page 103.

The ServiceComponent stanza

The `ServiceComponent` stanza identifies the name and path of the module containing the code for that component.

There can be more than one ServiceComponent stanza for each service, but each ServiceComponent stanza must match the corresponding Service stanza.

In MQSeries for UNIX systems, the authorization service stanza is present by default, and the associated component, the OAM, is active.

Service=*service_name*
Specifies the name of the required service. This name must match the value specified on the `Name` attribute of the `Service` stanza.

Name=*component_name*
Specifies the descriptive name of the service component. This name must be unique, and must contain only those characters that are valid for the names of MQSeries objects (for example, queue names). This name occurs in operator messages generated by the service. It is recommended, therefore, that this name begins with a company trademark or similar distinguishing string.

Module=*module_name*
Specifies the name of the module to contain the code for this component.

Note: Specify a full path name.

ComponentDataSize=*size*
Specifies the size, in bytes, of the component data area passed to the component on each call. Specify zero if no component data is required.

For more information about installable services and components, see Chapter 11, "Installable services and components" in the *MQSeries Programmable System Management* book.

The Log stanza

The `Log` stanza specifies the log attributes for a particular queue manager. By default, these are inherited from the settings specified in the `LogDefaults` stanza in the mqs.ini file when the queue manager is created.

Only change attributes of this stanza if this particular queue manager needs to be configured differently from your other ones.

The values specified on the attributes in the qm.ini file are read when the queue manager is started. The file is created when the queue manager is created.

For information about calculating log sizes, see "Calculating the size of the log" on page 219.

Note: The limits given in the following parameter list are limits set by MQSeries. Operating system limits may reduce the maximum possible log size.

LogPrimaryFiles=3|*2-62*
Primary log files are the log files allocated during creation for future use.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect may not be immediate.

LogSecondaryFiles=2|*1-61*
Secondary log files are the log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect may not be immediate.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

In MQSeries for UNIX systems, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages 64 and the maximum is 16 384.

In MQSeries for OS/2 Warp and MQSeries for Windows NT, the default number of log file pages is 256, giving a log file size of 1 MB. The minimum number of log file pages is 32 and the maximum is 4095.

Note: The size of the log files specified during queue manager creation cannot be changed for a queue manager.

LogType=CIRCULAR|LINEAR

The `LogType` attribute defines the type of logging to be used by the queue manager. However, you cannot change the type of logging to be used once the queue manager has been created. Refer to the description of the LogType attribute in "The LogDefaults stanza" on page 132 for information about creating a queue manager with the type of logging you require.

CIRCULAR

Set this value if you want to start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See "Circular logging" on page 215 for a fuller explanation of circular logging.

LINEAR

Set this value if you want both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See "Linear logging" on page 215 for a fuller explanation of linear logging.

LogBufferPages=17|*4-32*

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 minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

The default number of buffer pages is 17, equating to 68 KB.

The value is examined when the queue manager is 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.

LogPath=*directory_name*

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 in MQSeries for UNIX systems.
- `<prefix>\LOG` for MQSeries for OS/2 Warp and MQSeries for Windows NT, where <prefix> is the value as defined in the Prefix attribute of the corresponding QueueManager stanza in the mqs.ini file. On many machines, this is `C:\MQM\LOG`

You can specify the name of a directory on the **crtmqm** command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the **crtmqm** command, the value of the `LogDefaultPath` attribute in the mqs.ini file is used.

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

The RestrictedMode stanza

The `RestrictedMode` stanza is set by the **-g** option on the **crtmqm** command. You **must not** change this stanza after the queue manager has been created. If you do not use the -g option, the stanza is not created in the qm.ini file.

ApplicationGroup
Specifies the name of the group whose members are allowed to :

- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories.

This applies to MQSeries for AIX, Sun Solaris, and HP-UX systems only.

The XAResourceManager stanza

The `XAResourceManager` stanza specifies the resource managers to be involved in global units of work coordinated by the queue manager.

One `XAResourceManager` stanza is required in qm.ini for each instance of a resource manager participating in global units of work; no default values are supplied via mqs.ini.

See “Database coordination” on page 176 for more information about adding `XAResourceManager` attributes to qm.ini.

Name=*name* (mandatory)
This attribute identifies the resource manager instance.

The `Name` value can be up to 31 characters in length and must be unique within qm.ini. You can use the name of the resource manager as defined in its XA-switch structure. However, if you are using more than one instance of the same resource manager, you must construct a unique name for each instance. You could ensure uniqueness by including the name of the database in the `Name` string, for example.

MQSeries uses the `Name` value in messages and in output from the **dspmqtrn** command.

You are recommended not to change the name of a resource manager instance, or to delete its entry from qm.ini once the associated queue manager has started and the resource-manager name is in effect.

SwitchFile=*name* **(mandatory)**
This attribute specifies the fully-qualified name of the load file containing the resource manager's XA switch structure.

XAOpenString=*string* **(optional)**
This attribute specifies the string of data to be passed to the resource manager's xa_open entry point. The contents of the string depend on the resource manager itself. For example, the string could identify the database that this instance of the resource manager is to access. For more information about defining this attribute, see:

- "Adding the XAResourceManager stanza for DB2" on page 184
- "Adding XAResourceManager configuration information for Oracle" on page 189
- "Adding XAResourceManager configuration information for Sybase" on page 199

and consult your resource manager documentation for the appropriate string.

XACloseString=*string* **(optional)**
This attribute specifies the string of data to be passed to the resource manager's xa_close entry point. The contents of the string depend on the resource manager itself. For more information about defining this attribute, see:

- "Adding the XAResourceManager stanza for DB2" on page 184
- "Adding XAResourceManager configuration information for Oracle" on page 189
- "Adding XAResourceManager configuration information for Sybase" on page 199

and consult your database documentation for the appropriate string.

ThreadOfControl=THREAD|PROCESS
This attribute is mandatory for MQSeries for OS/2 Warp and MQSeries for Windows NT. The value set on the ThreadOfControl attribute is used by the queue manager for serialization purposes when it needs to call the resource manager from one of its own multithreaded processes.

THREAD
Means that the resource manager is fully "thread aware". In a multithreaded MQSeries process, XA function calls can be made to the external resource manager from multiple threads at the same time.

PROCESS
Means that the resource manager is not "thread safe". In a multithreaded MQSeries process, only one XA function call at a time can be made to the resource manager.

The `ThreadOfControl` entry does not apply to XA function calls issued by the queue manager in a multithreaded application process. In general, an application that has concurrent units of work on different threads requires this mode of operation to be supported by each of the resource managers.

The Channels stanza

The Channels stanza contains information about the channels.

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

MaxActiveChannels=*MaxChannels_value*
This attribute specifies the maximum number of channels allowed to be active at any time. The default is the value specified on the MaxChannels attribute.

MaxInitiators=3|*number*
This attribute specifies the maximum number of initiators.

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

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

STANDARD
Channels connect using STANDARD.

AdoptNewMCA=NO|SVR|SNDR|RCVR|CLUSRCVR|ALL|FASTPATH
If MQSeries receives a request to start a channel but finds that an amqcrsta process already exists for the same channel, the existing process must be stopped before the new one can start. The AdoptNewMCA attribute allows you to control the termination of an existing process and the startup of a new one for a specified channel type.

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

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

You specify one or more values, separated by commas or blanks, from the following list:

NO
The AdoptNewMCA feature is not required. This is the default.

SVR
Adopt server channels

SNDR
Adopt sender channels

RCVR
Adopt receiver channels

CLUSRCVR
Adopt cluster receiver channels

ALL
Adopt all channel types, except for FASTPATH channels

FASTPATH
Adopt the channel if it is a FASTPATH channel. This happens only if the appropriate channel type is also specified, for example, AdoptNewMCA=RCVR,SVR,FASTPATH

Attention!

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

AdoptNewMCATimeout=60|1—3600
This attribute specifies the amount of time, in seconds, that the new process should wait for the old process to end. Specify a value, in seconds, in the range 1—3600. The default value is 60.

AdoptNewMCACheck=QM|ADDRESS|NAME|ALL
The `AdoptNewMCACheck` attribute allows you to specify the type checking required when enabling the `AdoptNewMCA` attribute. It is important for you to perform all three of the following checks, if possible, to protect your channels from being, inadvertently or maliciously, shut down. At the very least check that the channel names match.

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

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

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

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

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

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

The LU62, NETBIOS, TCP, and SPX stanzas

These stanzas specify network protocol configuration parameters. They override the default attributes for channels.

Note: Only attributes representing changes to the default values need to be specified.

LU62 (MQSeries for OS/2 Warp and MQSeries for Windows NT only)
The following attributes can be specified:

TPName
This attribute specifies the TP name to start on the remote site.

Library1=*DLLName 1*
This attribute specifies the name of the APPC DLL.

The default value for MQSeries for OS/2 Warp is APPC.

The default value for MQSeries for Windows NT is WCPIC32.

Library2=*DLLName2*
This attribute is the same as Library1, and applies if the code is stored in two separate libraries. The default value MQSeries for OS/2 Warp is ACSSVC.

The default value MQSeries for Windows NT is WCPIC32.

LocalLU
This is the name of the logical unit to use on local systems.

NETBIOS (MQSeries for OS/2 Warp and MQSeries for Windows NT only)
The following attributes can be specified:

LocalName=*name*
This attribute specifies the name that this machine will be known as on the LAN.

AdapterNum=0|*adapter_number*
This attribute specifies the number of the LAN adapter. The default is adapter 0.

NumSess=1|*number_of_sessions*
This attribute specifies the number of sessions to allocate. The default is 1.

NumCmds=1|*number_of_commands*
This attribute specifies the number of commands to allocate. The default is 1.

NumNames=1|*number_of_names*
This attribute specifies the number of names to allocate. The default is 1.

Library1=*DLLName1*
This attribute specifies the name of the NetBIOS DLL. The default value for MQSeries for OS/2 Warp is ACSNETB.

The default value for MQSeries for Windows NT is NETAPI32.

Library2=*DLLName2* **(MQSeries for OS/2 Warp only)**
This attribute specifies the same value as Library1, if the code is in two separate libraries. There are no defaults for this attribute.

TCP

The following attributes can be specified:

Port=1414|*port_number*

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

Library1=*DLLName1* **(MQSeries for OS/2 Warp and MQSeries for Windows NT only)**

Use this attribute to specify the name of the TCP/IP sockets DLL.

The default for MQSeries for OS/2 Warp is SO32DLL.

The default for MQSeries for Windows NT is WSOCK32.

Library2=*DLLName 2* **(MQSeries for OS/2 Warp only)**

Same as Library1 if the code is stored in two separate libraries.

The default for MQSeries for OS/2 Warp is TCP32DLL.

KeepAlive=YES|NO

Use this attribute to switch the KeepAlive function on or off. KeepAlive=YES causes TCP/IP to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

ListenerBacklog=number

When receiving on TCP/IP, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the TCP/IP port for the listener to accept the request. The default listener backlog values are shown in Table 10.

Table 10. Default outstanding connection requests (TCP)

Platform	Default ListenerBacklog value
OS/390	255
OS/2 Warp	10
Windows NT Server	100
Windows NT Workstation	5
AS/400	255
Sun Solaris	100
HP-UX	20
AIX V4.2 or later	100
AIX V4.1 or earlier	10
All other platforms	5

If the backlog reaches the values shown in Table 10, the TCP/IP connection is rejected and the channel will not be able to start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

The ListenerBacklog attribute allows you to override the default number of outstanding requests for the TCP/IP listener.

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

SPX (MQSeries for OS/2 Warp and MQSeries for Windows NT only)

The following attributes can be specified:

Socket=5E86|*socket_number*

This attribute specifies the SPX socket number in hexadecimal notation. The default is X'5E86'.

BoardNum=0|*adapter_number*

This attribute specifies the LAN adapter number. The default is adapter 0.

KeepAlive=YES|NO

Use this attribute to switch the KeepAlive function on or off.

KeepAlive=YES causes SPX to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

LibraryName1=*DLLName1*

This attribute specifies the name of the SPX DLL.

The default for MQSeries for OS/2 Warp is IPXCALLS.DLL.

The default for MQSeries for Windows NT is WSOCK32.DLL.

LibraryName2=*DLLName2*

This attribute specifies the same value as LibraryName 1 if the code is held in two separate libraries.

The default for MQSeries for OS/2 Warp is SPXCALLS.DLL.

ListenerBacklog=number

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered a to be a *backlog* of requests waiting on the SPX socket for the listener to accept the request. The default listener backlog values are shown in Table 11.

Table 11. Default outstanding connection requests (SPX)

Platform	Default ListenerBacklog value
OS/2 Warp	10
Windows NT Server	100
Windows NT Workstation	5

If the backlog reaches the values shown in Table 10 on page 145, the SPX connection is rejected and the channel will not be able to start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

The ListenerBacklog attribute allows you to override the default number of outstanding requests for the SPX listener.

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

The ExitPath stanza

The ExitPath stanza applies to: Version 5.1 of:

- MQSeries for AIX
- HP-UX
- OS/2 Warp
- Sun Solaris
- Windows NT

ExitDefaultPath=*string*

The ExitDefaultPath attribute specifies the location of:

- Channel exits for clients
- Channel exits and data conversion exits for servers

The exit path is read from the ClientExitPath stanza in the mqs.ini file for clients and from this (ExithPath) stanza for servers.

For MQSeries for Windows NT, ClientExithPath and ExitPath information is in the Windows NT Registry.

The UDP stanza

The UDP stanza can be used to tailor User Datagram Protocol (UDP) support on your MQSeries system and is applicable to MQSeries AIX 5.1 systems only. UDP is part of the Internet suite of protocols and may be used as an alternative to TCP/IP.

You can use UDP to send message data between MQSeries for Windows Version 2.02 systems (that is with CSD 2 installed) and MQSeries for AIX Version 5.1 server systems.

A sample qm.ini file is shipped in the MQM\QMGRS\ directory. To use it, copy it to the sub-directory for the queue manager and edit it as required, using the following attribute descriptions to guide you.

ACKREQ_TIMEOUT=5|1–30 000

The request for acknowledgement timeout attribute, ACKREQ_TIMEOUT, specifies the time, in seconds, that the internal state machines will wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1—30 000.

ACKREQ_RETRY=60|1–30 000

The request for acknowledgment retry attribute, ACKREQ_RETRY, specifies the number of times that the internal state machines will resend protocol datagrams before giving up completely and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1—30 000.

CONNECT_TIMEOUT=5|1–30 000
The connect request timeout attribute, CONNECT_TIMEOUT, specifies the time, in seconds, that the internal state machines will wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1—30 000.

CONNECT_RETRY=60|1–30 000
The connect request retry attribute, CONNECT_RETRY, specifies the number of times that the internal state machines will resend protocol datagrams before giving up completely and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1—30 000.

ACCEPT_TIMEOUT=5|1–30 000
The accept connection timeout attribute, ACCEPT_TIMEOUT, specifies the time, in seconds, that the internal state machines will wait for a protocol datagram before assuming that the datagram has been lost and retrying. The default is 5 but you can change this to a value in the range 1–30 000.

ACCEPT_RETRY=60|1–30 000
The accept connection retry attribute, ACCEPT_RETRY, specifies the number of times that the internal state machines will resend protocol datagrams before giving up completely and causing a channel to close. (All the counts are reset to zero after success and thus are not cumulative.)

The default is 60 but you can change this to a value in the range 1—30 000.

DROP_PACKETS=0|1–30 000
The DROP_PACKETS attribute tests for the robustness of the protocols against lost datagrams. Changing the value to something other than 0 causes datagrams to be thrown away and the protocol will cause them to be resent. Therefore, you are advised not to change the value of this attribute to anything other than 0 for normal usage.

BUNCH_SIZE=8|1–30 000
The BUNCH_SIZE attribute specifies the number of datagrams that are sent before an acknowledgement datagram is sent from the receiving node. The default is 8.

Changing the default to a value higher than 8 may reduce the number of datagrams sent but may also affect other aspects of performance. Without knowing the details of the network involved, it is difficult to suggest exactly how to vary the values on this attribute, but a good rule of thumb is probably that the longer the network delay, the larger the value of BUNCH_SIZE should be for optimum performance.

PACKET_SIZE=2048|512–8192
The PACKET_SIZE attribute specifies the maximum size of UDP datagrams sent over the IP network. Some networks may have a limit as low as 512 bytes. The default value of 2048 appears to be successful most of the time. However, if you experience problems with this value, you can slowly increase it from 512 until you find your own optimum value.

PSEUDO_ACK=NO|YES
Set the PSEUDO_ACK attribute to YES if you want the datagram that is about to be sent to be modified so that it requests the remote end of the link to send an “information” datagram back to indicate that the node can be reached.

PSEUDO_ACK=YES must be set at both the remote and local ends of the channel.

The default is NO.

The Transport stanza

The `Transport` stanza is used to tailor User Datagram Protocol (UDP) support on your MQSeries system and must be coded in conjunction with the UDP stanza above.

RETRY_EXIT=exitname

The RETRY_EXIT attribute specifies the name of the library that contains the retry exit. The retry exit allows your application to suspend data being sent on a channel when communication is not possible.

For Windows systems, the retry exit name takes the form `xyz.DLL(myexit)` while for AIX systems, the retry exit name takes the form `xyz(myexit)`.

For more information about the retry exit, see “The retry exit” on page 376.

Example mqs.ini and qm.ini files for MQSeries for OS/2 Warp

The mqs.ini file shown in Figure 11 and the qm.ini files shown in Figure 12 on page 151, Figure 13 on page 152, and Figure 14 on page 153, were created by:

1. Installing MQSeries for OS/2 Warp and specifying:
   - G:\MQM as the file directory, which puts program files onto the G: drive
   - M:\MQM as the work directory, which puts queues onto the M: drive.
2. Editing the M:\MQM\MQS.INI file to change the value on the `LogDefaultPath` attribute to

   ```
   R:\MQM\LOG
   ```

   This puts the log files to the R: drive rather than the M: drive.
3. Creating three queue managers using the following commands:
   - `crtmqm -q firstqm`
   - `crtmqm secondqm`
   - `crtmqm -lf 1024  -lp 10  -ls 5 thirdqm`

Resultant mqs.ini file (MQSeries for OS/2 Warp)

Path M:\MQM\MQS.INI

```
#********************************************************************#
#*  Module Name: mqs.ini                                            #
#*  Type       : MQSeries Machine-Wide Configuration File           #
#*  Function   : Define MQSeries resources for an entire machine    #
#*                                                                  #
#********************************************************************#
#*  Notes                                                           #
#*  1) This is the installation time default configuration          #
#*                                                                  #
#********************************************************************#
AllQueueManagers:
  #******************************************************************#
  #* The path to the qmgrs directory, below which the queue manager #
  #* data is stored                                                 #
  #******************************************************************#
  DefaultPrefix=M:\MQM
  DefaultFilePrefix=G:\MQM

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=256
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=R:\MQM\LOG

QueueManager:
  Name=firstqm
  Prefix=M:\MQM
  Directory=firstqm

DefaultQueueManager:
  Name=firstqm

QueueManager:
  Name=secondqm
  Prefix=M:\MQM
  Directory=secondqm

QueueManager:
  Name=thirdqm
  Prefix=M:\MQM
  Directory=thirdqm
```

Figure 11. Example of an mqs.ini file for MQSeries for OS/2 Warp

Resultant qm.ini file for queue manager firstqm (MQSeries for OS/2 Warp)

Path M:\MQM\QMGRS\FIRSTQM\QM.INI

```
#*******************************************************************#
#*  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    #
#*                                                                 #
#*******************************************************************#
ExitPath:
  ExitsDefaultPath=M:\MQM\exits

Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=256
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=R:\MQM\LOG\firstqm\
```

Figure 12. Example of qm.ini file for queue manager firstqm

Resultant qm.ini file for queue manager secondqm (MQSeries for OS/2 Warp)

Path M:\MQM\QMGRS\SECONDQM\QM.INI

```
#********************************************************************#
#*  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     #
#*                                                                  #
#********************************************************************#
ExitPath:
  ExitsDefaultPath=M:\MQM\exits

Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=256
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=R:\MQM\LOG\secondqm\
```

Figure 13. Example of qm.ini file for queue manager secondqm

Resultant qm.ini file for queue manager thirdqm (MQSeries for OS/2 Warp)

Path M:\MQM\QMGRS\THIRDQM\QM.INI

```
#*******************************************************************#
#*  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    #
#*                                                                 #
#*******************************************************************#
ExitPath:
  ExitsDefaultPath=M:\MQM\exits

Log:
  LogPrimaryFiles=10
  LogSecondaryFiles=5
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=R:\MQM\LOG\thirdqm\
```

Figure 14. Example of qm.ini file for queue manager thirdqm

Example mqs.ini and qm.ini files for MQSeries for UNIX systems

Figure 15 shows an example of an mqs.ini file in MQSeries for UNIX systems.

```
#********************************************************************#
#* 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
```

Figure 15. Example of an MQSeries configuration file for UNIX systems

Figure 16 on page 155 shows how groups of attributes might be arranged in a queue manager configuration file in MQSeries for UNIX systems.

```
#********************************************************************#
#* 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     *#
#*                                                                  *#
#********************************************************************#
ExitPath:
   ExitsDefaultPath=/var/mqm/exits

Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=mqmtop/bin/amqzfu.o
   ComponentDataSize=0

Service:
   Name=NameService
   EntryPoints=5

ServiceComponent:
   Service=NameService
   Name=MQSeries.DCE.name.service
   Module=mqmtop/lib/amqzfa
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=17
   LogPath=/var/mqm/log/saturn!queue!manager/

XAResourceManager:
  Name=DB2 Resource Manager Bank
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB
  XACloseString=
  ThreadOfControl=PROCESS
```

Figure 16 (Part 1 of 2). Example queue manager configuration file for MQSeries for UNIX systems

```
CHANNELS:
  MaxChannels = 20          ; Maximum number of Channels allowed.
                            ; Default is 100.
  MaxActiveChannels = 10    ; Maximum number of Channels allowed to be
                            ; active at any time.  The default is the
                            ; value of MaxChannels.

TCP:                        ; TCP/IP entries.
  KeepAlive = Yes           ; Switch KeepAlive on
```

Figure 16 (Part 2 of 2). Example queue manager configuration file for MQSeries for UNIX systems

Notes for examples:

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.

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

Chapter 12. 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.

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 contains the following sections:

Invoking the DLQ handler

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

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

```
runmqdlq ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER <qrule.rul
```

- In the rules table. For example:

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

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

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

The **runmqdlq** command takes its input from `stdin`; you associate the rules table with **runmqdlq** by redirecting `stdin` from the rules table.

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 "runmqdlq (Run dead-letter queue handler)" on page 318.

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.)

In MQSeries for UNIX systems, the source of amqsdlq is supplied in the directory:

```
mqmtop/samp/dlq
```

and the compiled version is supplied in the directory:

```
mqmtop/samp/bin
```

In MQSeries for OS/2 Warp and Windows NT, the source of amqsdlq is supplied in the directory:

```
c:\mqm\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:

```
c:\mqm\tools\c\samples\bin
```

The DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table. Note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName***|'_')**
Allows you to name the DLQ you want to process:

1. If you specify an INPUTQ value as a parameter to the **runmqdlq** command, this overrides any INPUTQ value in the rules table.
2. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command, but you *do* specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If no DLQ is specified or you specify INPUTQ(' ') in the rules table, the name of the DLQ belonging to the queue manager whose name is supplied as a parameter to the **runmqdlq** command is used.
4. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command or as a value in the rules table, the DLQ belonging to the queue manager named on the INPUTQM keyword in the rules table is used.

INPUTQM (*QueueManagerName***|'_')**
Allows you to name the queue manager that owns the DLQ named on the INPUTQ keyword:

1. If you specify an INPUTQM value as a parameter to the **runmqdlq** command, this overrides any INPUTQM value in the rules table.
2. If you do not specify an INPUTQM value as a parameter to the **runmqdlq** command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval***|60)**
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 invoke it again by means of triggering. For more information about triggering, see Chapter 14, "Starting MQSeries applications using triggers" in the *MQSeries Application Programming Guide*.

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 17 shows an example rule from a DLQ handler rules table.

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

Figure 17. 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.

All keywords that you can use on a rule are described in the remainder of this section. Note the following:

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

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

The pattern-matching keywords

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

APPLIDAT (*ApplIdentityData***|*****)**

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.

DISCARD Causes the message to be deleted from the DLQ.

IGNORE Causes the message to be left on the DLQ.

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

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

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.

  For example:

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

results in 'ABCD', and

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

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in quoted values are not considered significant. For example, these parameters are valid:

    ```
    FORMAT('ABC')   3 significant characters
    FORMAT(ABC)     3 significant characters
    FORMAT('A')     1 significant character
    FORMAT(A)       1 significant character
    FORMAT(' ')     1 significant character
    ```

    These parameters are invalid because they contain no significant characters:

    ```
    FORMAT('')
    FORMAT( )
    FORMAT()
    FORMAT
    ```

  - Wildcard characters are supported: you can use the question mark (?) 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.

Notes:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule 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 “AMQ8000-AMQ8499 MQSeries administration messages” in the *MQSeries Messages* book.) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.

4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table may not be recognized because the implementation of the validation of the rules table is designed to eliminate the generation of repetitive errors.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler could run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still 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

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

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

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

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

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

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

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

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

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

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

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

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
```

```
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

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

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

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

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

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

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

Chapter 13. Instrumentation events

You can use MQSeries instrumentation events to monitor the operation of queue managers. This chapter provides a short introduction to instrumentation events. For a more complete description, see Chapter 1, “Using instrumentation events to monitor queue managers” in the *MQSeries Programmable System Management* book.

What are instrumentation events?

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

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

Other conditions that can give rise to instrumentation events include:

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

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

Figure 18 on page 170 summarizes the production of an event message.

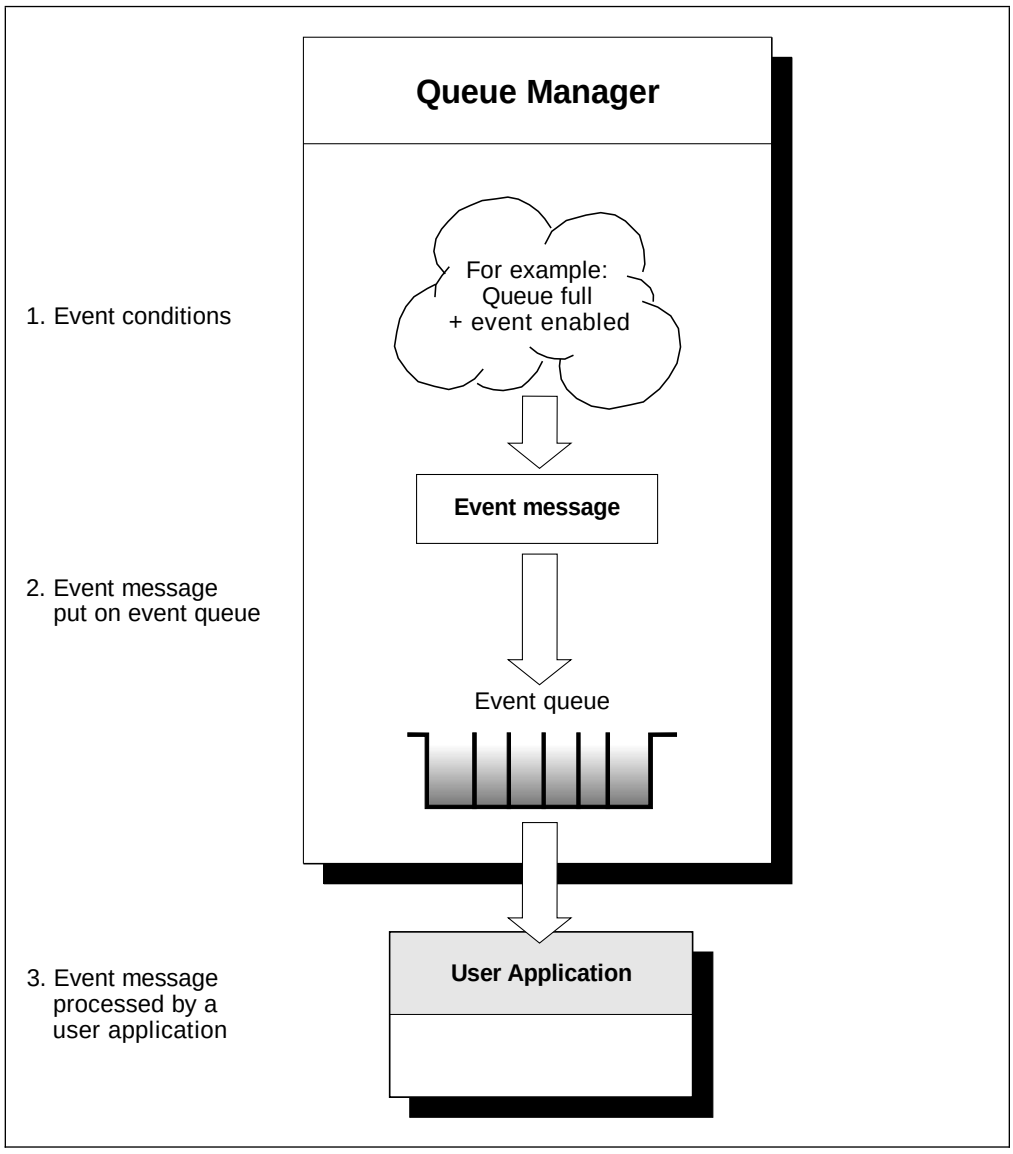


Figure 18. Understanding instrumentation events. When a queue manager detects that the conditions for an event have been met, it puts an event message on the appropriate event queue.

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

Why use events?

If you define your event queues as remote queues, you can put all the event queues on a single queue manager (for those nodes that support instrumentation events). You can then use the events generated to monitor a network of queue managers from a single node. Figure 19 on page 171 illustrates this.

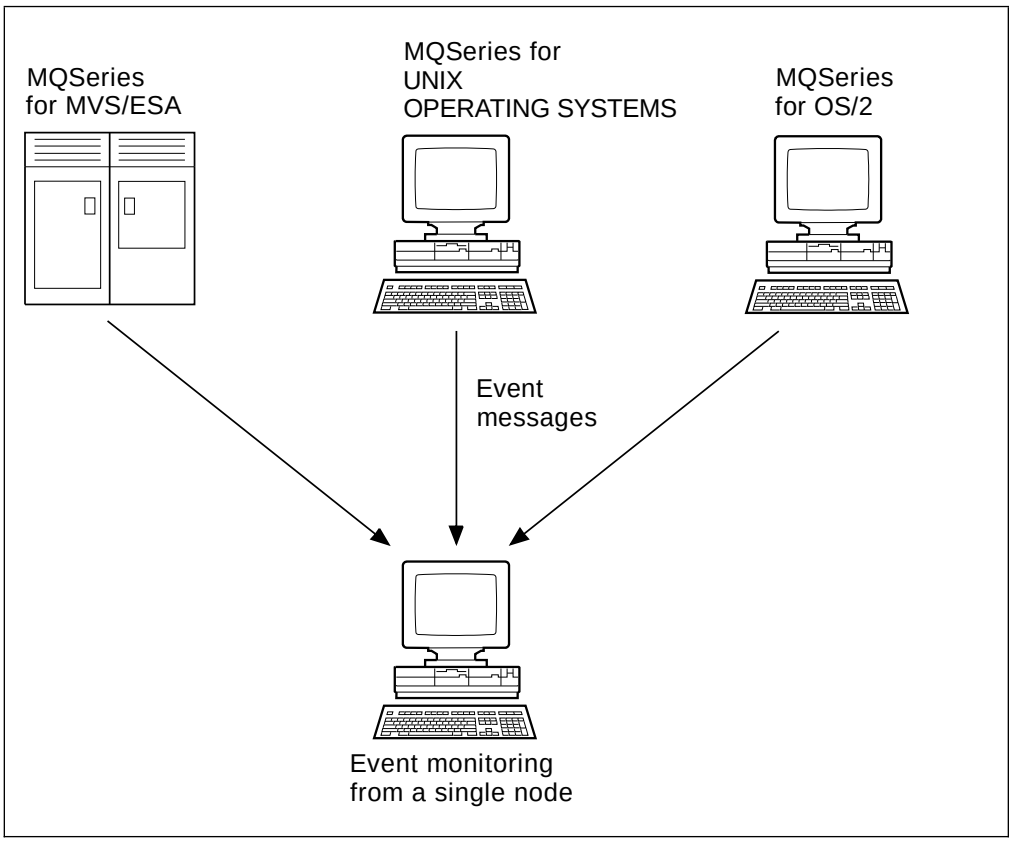


Figure 19. Monitoring queue managers across different platforms, on a single node

Types of event

MQSeries events are categorized as follows:

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

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

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

Trigger events

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

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

Event notification through event queues

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

- Gets the message from the queue.
- Processes the message to extract the event data. For a description of event message formats, see Chapter 4, “Event message reference” in the *MQSeries Programmable System Management* book.

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

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

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

Using triggered event queues

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

Enabling and disabling events

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

- MQSC commands. For more information, see Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual.
- PCF commands. For more information, see “Enabling and disabling events” in the *MQSeries Programmable System Management* manual.

- MQAI commands. For more information, see the *MQSeries Administration Interface Programming Guide and Reference* book.

Enabling an event depends on the category of the event:

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

Event messages

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

The format of event messages is similar to that of PCF response messages. The message data can be retrieved from them by user-written administration programs using the data structures described in Chapter 4, “Event message reference” in the *MQSeries Programmable System Management* manual.

Chapter 14. Transactional support

Chapter 13, “Committing and backing out units of work” in the *MQSeries Application Programming Guide* contains a complete introduction to the subject of this chapter. A brief introduction only is provided here.

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

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

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

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

In summary, queue manager resources can be updated as part of local or global units of work:

Local unit of work
Use local units of work when the only resources to be updated are those of the MQSeries queue manager. Updates are committed using the MQCMIT verb or backed out using MQBACK.

Global unit of work
Use global units of work when you also need to include updates to XA-compliant database managers. Here the coordination may be internal or external to the queue manager.

Queue manager coordination
Global units of work are started using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK. A two-phase commit process is used whereby XA-compliant resource managers such as DB2®, Oracle, and Sybase are firstly all asked to prepare to commit. Only if all are prepared successfully will they then be asked to commit. If any resource manager signals that it cannot prepare to commit, each will be asked to back out instead.

External coordination
Here the coordination is performed by an XA-compliant transaction manager such as IBM CICS or BEA Tuxedo. Units of work are started and committed under control of the transaction manager. The MQBEGIN, MQCMIT and MQBACK verbs are unavailable.

This chapter describes how to enable support for global units of work (support for local units of work does not need to be specifically enabled).

It contains these sections:

Database coordination

When the queue manager coordinates global units of work itself it becomes possible to integrate database updates within MQ units of work. That is, a mixed MQI and SQL application can be written, and the MQCMIT and MQBACK verbs can be used to commit or roll back the changes to the queues and databases together.

The queue manager achieves this using a two-phase commit protocol. When a unit of work is to be committed, the queue manager first asks each participating database manager whether it is prepared to commit its updates. Only if all of the participants, including the queue manager itself, are prepared to commit, are all of the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

Full recovery support is provided if the queue manager loses contact with any of the database managers during the commit protocol. If a database manager becomes unavailable while it is in doubt, that is, it has been called to prepare but has yet to receive a commit or back out decision, the queue manager remembers the outcome of the unit of work until it has been successfully delivered. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered over queue manager restart.

The MQI verb, MQBEGIN, must be used to denote units of work that are also to involve database updates. Chapter 13, "Committing and backing out units of work" in the *MQSeries Application Programming Guide* identifies sample programs that make MQSeries and database updates within the same unit of work.

The queue manager communicates with the database managers using the XA interface as described in *X/Open Distributed Transaction Processing: The XA Specification* (ISBN 1 872630 24 3). This means that the queue manager can communicate to database managers that also adhere to this standard. Such database managers are known as *XA-compliant* database managers.

Table 12 on page 177 identifies XA-compliant database managers that are supported by the MQSeries Version 5 products.

Table 12. XA-compliant relational databases			
MQSeries product	**DB2**	**Oracle**	**Sybase**
MQSeries for AIX	Yes	Yes	Yes
MQSeries for HP-UX	Yes	Yes	No
MQSeries for OS/2 Warp	Yes	No	No
MQSeries for Sun Solaris	Yes	Yes	Yes
MQSeries for Windows NT	Yes	No	Yes

Restrictions

The following restrictions apply to the database coordination support:

- The ability to coordinate database updates within MQSeries units of work is **not** supported in an MQI client application.
- The MQI updates and database updates must be made on the same queue manager server machine.
- The database server may reside on a different machine from the queue manager server. In this case, the database needs to be accessed via an XA-compliant client feature provided by the database manager itself.
- Although the queue manager itself is XA-compliant, it is not possible to configure another queue manager as a participant in global units of work. This is because only one connection at a time can be supported.

Database connections

An application that establishes a standard connection to the queue manager will be associated with a thread in a separate local queue manager agent process. When the application issues MQBEGIN then both it and the agent process will need to connect to the databases that are to be involved in the unit of work. The database connections are maintained while the application remains connected to the queue manager. This is an important consideration if the database only supports a limited number of users or connections.

One method of reducing the number of connections is for the application to use the MQCONNX call to request a fastpath binding. In this case the application and the local queue manager agent become the same process and consequently can share a single database connection. Before you do this, consult “Connecting to a queue manager using the MQCONNX call” in the *MQSeries Application Programming Guide* for a list of restrictions that apply to fastpath applications.

Configuring database managers

There are several tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager:

1. Create an *XA switch load file*[4] for the database manager.
2. Define the database manager in the queue manager's configuration file, qm.ini, or, for users of MQSeries for Windows NT Version 5.1 or later, the Windows NT Registry.

   Various items, including the name of the switch load file, must be defined in qm.ini or, for MQSeries for Windows NT Version 5.1 only, in the Windows NT Registry.

Creating switch load files

A sample makefile is shipped with each of the MQSeries Version 5.1 products which can be used to build switch load files for the supported database managers

This makefile, together with all the associated files required to build the switch load files, is installed in the following directories:

- For MQSeries for OS/2 Warp or Windows NT in the \mqm\tools\c\samples\xatm\ directory
- For MQSeries for UNIX systems, in the **mqmtop**/samp/xatm/ directory

Refer to your MQSeries installation documentation for more information about the installation procedure.

The sample source modules that are used to produce the switch load files all contain a single function called MQStart. When the switch load file is loaded, the queue manager calls this function and it returns the address of a structure called an XA switch. The switch load file is linked to a library provided by the database manager, which enables MQSeries to call that database manager.

The sample source modules used to build the switch load files are:

- For DB2, db2swit.c
- For Oracle, oraswit.c
- For Sybase, sybswit.c

Defining database managers

When you have created a switch load file for your database manager, you must specify its location to your queue manager. This is done in the queue manager's qm.ini file in the `XAResourceManager` stanza, or for users of MQSeries for Windows NT Version 5.1 or later, in the Windows NT Registry.

You need to add an `XAResourceManager` stanza for each database manager that your queue manager is going to coordinate. More complicated configurations involving multiple databases, or different database managers, are discussed in "Multiple database configurations" on page 200.

[4] An XA switch load file is a dynamically loaded object that enables the queue manager and the database manager to communicate with each other.

The attributes of the `XAResourceManager` stanza are as follows.

Name=name
User-chosen string that identifies the database manager instance.

The name is mandatory and can be up to 31 characters in length. It must be unique. It could simply be the name of the database manager, although to maintain its uniqueness in more complicated configurations it could, for example, also include the name of the database being updated.

The name that you choose should be meaningful because the queue manager uses it to refer to this database manager instance both in messages and in output when the **dspmqtrn** command is used.

Once you have chosen a name, **do not change this attribute**. Information about changing configuration information is given in "Changing configuration information" on page 205.

SwitchFile=name
This is the fully-qualified name of the database manager's XA switch load file. This is a mandatory attribute.

XAOpenString=string
This is a string of data that is passed in calls to the database manager's xa_open entry point. The format for this string depends on the particular database manager, but it should usually identify the name of the database that is to be updated.

This is an optional attribute; if it is omitted a blank string is assumed.

XACloseString=string
This is a string of data that is passed in calls to the database manager's xa_close entry point. The format for this string depends on the particular database manager.

This is an optional attribute; if it is omitted a blank string is assumed.

ThreadOfControl=THREAD|PROCESS
This attribute applies only to MQSeries for OS/2 Warp and MQSeries for Windows NT products, where it is mandatory. The *ThreadOfControl* value can be THREAD or PROCESS. The queue manager uses it for serialization purposes.

If the database manager is "thread-safe", the value for *ThreadOfControl* can be THREAD, and the queue manager can call the database manager from multiple threads at the same time.

If the database manager is not thread-safe, the value for *ThreadOfControl* should be PROCESS. The queue manager serializes all calls to the database manager so that only one call at a time is made from within a particular process.

See "The XAResourceManager stanza" on page 140 for fuller descriptions of these attributes.

"DB2 configuration" on page 180, "Oracle configuration" on page 186, and "Sybase configuration" on page 192 give more information about the specific tasks you need to perform to configure MQSeries with each of the supported database managers.

DB2 configuration

The minimum supported level of DB2 is DB2 Universal Database®, Version 5.0.

You need to perform the following tasks:

- Check the environment variable settings.
- Create the DB2 switch load file.
- Add XAResourceManager configuration information to the qm.ini file, or, for MQSeries for Windows NT only, to the Windows NT Registry.
- Change DB2 configuration parameters if necessary.

Checking the environment variable settings

Ensure that your DB2 environment variables are set for queue manager processes as well as in your application processes. In particular, you must always set the DB2INSTANCE environment variable ***before*** you start the queue manager. The DB2INSTANCE environment variable identifies the DB2 instance containing the DB2 databases that are being updated.

Creating the DB2 switch load file

The easiest method for creating the DB2 switch load file is to use the sample file xaswit.mak. The source code used to create the DB2 switch on most platforms is shown in Figure 20. The source for db2swit.c for Windows NT is different; it is shown in Figure 21.

```
#include <cmqc.h>
#include "xa.h"

extern struct xa_switch_t db2xa_switch;

struct xa_switch_t * MQENTRY MQStart(void)
{
   return(&db2xa_switch);
}
```

Figure 20. Source code for db2swit.c for platforms other than Windows NT

```
#include <cmqc.h>
#include "xa.h"

extern __declspec(dllimport) struct xa_switch_t db2xa_switch;

struct xa_switch_t * MQENTRY MQStart(void)
{
   return(&db2xa_switch);
}
```

Figure 21. Source code for db2swit.c on Windows NT (Microsoft Visual C++-specific)

The xa.h header file that is supplied with MQSeries is installed (for MQSeries for OS/2 Warp or MQSeries for Windows NT) in the \mqm\tools\c\samples\xatm directory, or (for MQSeries for UNIX systems) in the **mqmtop**/samp/xatm directory.

Creating the DB2 switch load file on OS/2

To create the DB2 switch load file on OS2, db2swit.c must be compiled and linked against db2api.lib. The DEF file shown in Figure 22 is needed to produce the DLL:

```
LIBRARY DB2SWIT

CODE SHARED    LOADONCALL
DATA NONSHARED MULTIPLE

EXPORTS
   MQStart                  @1
```

Figure 22. Source code for db2swit.def on OS/2

To create the DLL:

1. Create a directory into which you want the switch file to be built. The switch file must be defined to MQSeries as a fully-qualified name so the DLL does not need to be built into a directory within LIBPATH.
2. Copy the following files from \mqm\tools\c\samples\xatm into your new directory:
   - xa.h
   - db2swit.c
   - db2swit.def
   - xaswit.mak
3. Use the source code for xaswit.mak on OS/2 shown in Figure 23 to build the switch load file.

```
CFLAGS=/c /Ss /Gm /Ge- /Q /Sp1
LFLAGS=/NOFREE /noi /align:16 /exepack

.SUFFIXES: .c .obj

db2swit.dll: db2swit.obj db2swit.def {$(LIB)}db2api.lib

.obj.dll:
    $(CC) /B"$(LFLAGS)" /Fe $@ $**

.c.obj:
    $(CC) $(CFLAGS) $*.c
```

Figure 23. Makefile for DB2 switch on OS/2

4. Issue an `nmake -f xaswit.mak db2swit.dll` command to make the DB2 switch load file, db2swit.dll.

Creating the DB2 switch load file on Windows NT

To create the DB2 switch load file on Windows NT, db2swit.c must be compiled and linked against db2api.lib. The DEF file shown in Figure 24 is needed to produce the DLL:

```
LIBRARY DB2SWIT

EXPORTS
   MQStart
```

Figure 24. Source code for db2swit.def on Windows NT

To create the DLL:

1. Create a directory into which you want the switch file to be built. The switch file must be defined to MQSeries as a fully-qualified name so the DLL does not need to be built into a directory within PATH.
2. Copy the following files from \mqm\tools\c\samples\xatm into your new directory:
   - xa.h
   - db2swit.c
   - db2swit.def
   - xaswit.mak
3. Use the source code shown in Figure 25, which forms part of xaswit.mak on Windows NT, to build the switch load file:

```
!include <ntwin32.mak>

db2swit.lib db2swit.exp: $*.obj $*.def
                         $(implib) -machine:$(CPU) \
                         -def:$*.def $*.obj

db2swit.dll: $*.obj $*.def $*.exp
             $(link) $(dlllflags)    \
             -base:0x1C000000        \
             $*.exp $*.obj           \
             $(conlibsdll) db2api.lib

.c.obj:
    $(cc) $(cflags) $(cvarsdll) $*.c
```

Figure 25. Makefile for DB2 switch on Windows NT

4. Issue an `nmake -f xaswit.mak db2swit.dll` command to make the DB2 switch load file, db2swit.dll, using the Microsoft Visual C++ compiler.

Creating the DB2 switch load file on UNIX systems

To create the DB2 switch load file on UNIX systems, db2swit.c must be compiled and linked against libdb2.

To build the switch load file:

1. Create a directory into which the DB2 load file, db2swit, will be built.
2. Copy the following files from **mqmtop**/samp/xatm into this new directory:
   - xa.h
   - db2swit.c
   - xaswit.mak
3. Use the source shown in Figure 26, which forms part of xaswit.mak on AIX, to build the DB2 switch file on AIX.

```
DB2LIBS=-l db2
DB2LIBPATH=-L /usr/lpp/db2_05_00/lib

db2swit:
        $(CC) -e MQStart $(DB2LIBPATH) $(DB2LIBS) -o $@ db2swit.c
```

Figure 26. Makefile for DB2 switch on AIX

4. Use the source shown in Figure 27, which forms part of xaswit.mak on Sun Solaris, to build the DB2 switch file on Sun Solaris.

```
DB2LIBS=-l db2
DB2LIBPATH=-L /opt/IBMdb2/V5.0/lib -R /opt/IBMdb2/V5.0/lib

db2swit:
        $(CC) -G -e MQStart $(DB2LIBPATH) $(DB2LIBS) -o $@ db2swit.c
```

Figure 27. Makefile for DB2 switch on Sun Solaris

5. Use the source shown in Figure 28, which forms part of xaswit.mak on HP-UX, to build the DB2 switch file on HP-UX systems.

```
DB2LIBS=-l db2 -l cl
DB2LIBPATH=-L /opt/IBMdb2/V5.0/lib +b /opt/IBMdb2/V5.0/lib

db2swit: db2swit.c
        $(CC) -c -Ae +z db2swit.c

        ld -b -e MQStart $(DB2LIBPATH) $(DB2LIBS) -o db2swit db2swit.o
```

Figure 28. Makefile for DB2 switch on HP-UX

6. Issue a `make -f xaswit.mak db2swit` command to build the DB2 switch load file using the sample makefile.

Adding the XAResourceManager stanza for DB2

The next step is to modify the configuration information for the queue manager to define DB2 as a participant in global units of work in the qm.ini file or, for MQSeries for Windows NT only, in the Windows NT Registry.

Add an `XAResourceManager` stanza with the following attributes:

Name=name
This attribute is mandatory. Choose a suitable name for this participant; you could include the name of the database being updated.

SwitchFile=name
This attribute is mandatory. Specify the fully-qualified name of the DB2 switch load file.

XAOpenString=string
The XA open string for DB2 must be of the following format:

```
database_alias<,username,password>
```

where:

- `database_alias` is the name of the database, unless you have explicitly cataloged an alias name after the database was created in which case specify the alias instead.

  The following two parameters are optional. They provide alternative authentication information to the database if it was set up with **authentication=server**.

- `username` specifies a user name defined to DB2.
- `password` is the password for the specified user ID.

  See “Security considerations” on page 200 for more information about security.

XACloseString=string
DB2 does not require an XA close string.

ThreadOfControl=THREAD|PROCESS
DB2 is not thread-aware so specify *PROCESS*.

For fuller descriptions of each of these attributes, see “The XAResourceManager stanza” on page 140.

Figure 29, shows some sample XAResourceManager entries where the database to be updated is called `MQBankDB`, this name being specified as the *XAOpenString*.

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=c:\user\dll\db2swit.dll
  XAOpenString=MQBankDB
  ThreadOfControl=PROCESS
```

Figure 29. Sample XAResourceManager entry for DB2 on OS/2 and Windows NT

Figure 30 on page 185 is a UNIX sample. It is assumed that the DB2 switch load file was copied to the /usr/bin directory after it had been created:

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB
```

Figure 30. Sample XAResourceManager entry for DB2 on UNIX platforms

Changing DB2 configuration parameters

Perform each of the following steps to each DB2 database that is being coordinated by the queue manager.

- **Database privileges**
  The mqm user ID must be authorized to connect to the DB2 database so that the queue manager can connect to DB2 from within its own processes.

  For example, to give the mqm user ID connect authority to the MQBankDB database the following commands shown in Figure 31 could be used:

```
db2 connect to MQBankDB

db2 grant connect on database to user mqm
```

Figure 31. Sample commands to give connect user ID authority to MQBANKDB

  See “Security considerations” on page 200 for more information about security.

- **tp_mon_name parameter**

  For DB2 for OS/2 and DB2 for Windows NT only, the *TP_MON_NAME* configuration parameter must be updated to name the DLL that DB2 uses to call the queue manager for dynamic registration.

  This can be achieved using a `db2 update dbm cfg using TP_MON_NAME mqmax` command.

  This names MQMAX.DLL as the library that DB2 uses to call the queue manager. This must be present in a directory within LIBPATH for OS/2, or PATH for Windows NT.

- **maxappls parameter**
  You may need to review your setting for the *maxappls* parameter, which limits the maximum number of applications that can be connected to a database.

  Refer to “Database connections” on page 177.

Oracle configuration

You need to perform the following tasks:

- Check Oracle level and apply patches if you have not already done so.
- Check environment variable settings.
- Enable Oracle XA support.
- Create the Oracle switch load file.
- Add XAResourceManager configuration information to the qm.ini file, or, for MQSeries for Windows NT only, to the Windows NT Registry.
- Change the Oracle configuration parameters, if necessary.

Minimum supported levels for Oracle and applying patches

- The minimum supported level of Oracle on AIX is 7.3.2.1.
- The minimum supported level of Oracle on HP-UX is 7.3.2.3.
- The minimum supported level of Oracle on Sun Solaris is 7.3.2.3.
- You need to install Oracle patches 437448 and 441647.

Checking the environment variable settings

Ensure that your Oracle environment variables are set for queue manager processes as well as in your application processes. In particular, the following environment variables should always be set **prior** to starting the queue manager:

ORACLE_HOME Is the Oracle home directory

ORACLE_SID Is the Oracle SID being used

Enabling Oracle XA support

You need to ensure that Oracle XA support is enabled. In particular, an Oracle shared library must have been created; this happens during installation of the Oracle XA library. On Oracle7, you may be prompted with:

```
Some TP Monitors require a shared version of the ORACLE7 libraries.
Do you want to install a shared version of the libraries?
```

Make sure you answer `Yes` to this prompt. This creates a shared library called libclntsh in the $ORACLE_HOME/lib directory.

During installation of Oracle8, the library is built automatically. You are not prompted as described for Oracle7 above.

If you need to rebuild the library, enter the appropriate command while you are logged on as an Oracle administrator:

- For Oracle7:

```
cd $ORACLE_HOME/rdbms/lib
make -f clntsh.mk libclntsh
```

- For Oracle8:

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk client_sharedlib
```

For more information, refer to:

- The *Oracle7 Administrator's Reference for UNIX* publication.
- The *Oracle8 Administrator's Reference* publication appropriate to your platform.

The queue manager loads the XA switch when it starts up. The platform-specific environment variables (LIBPATH for AIX, LD_LIBRARY_PATH for Sun Solaris, and SHLIB_PATH for HP-UX) are not passed to the queue manager processes from the shell in which strmqm is called. Therefore, another method must be used so that the shared objects can be located during start up of the queue manager. You can do this either by:

- Imbedding the path to libclntsh in the XA switch load file when it is built. This is the recommended method.
- Providing a symbolic link to libclntsh from /usr/lib. (/usr/lib is, by default, searched for shared objects if none are found in the paths included by the first method.)

The example makefiles in "Creating the Oracle switch load file on UNIX systems" on page 188, are written assuming that you have **not** placed a symbolic link to libclntsh in /usr/lib. If you wish to use a symbolic link, you are free to remove the relevant switches from these sample makefiles (-R for Sun Solaris or +b for HP-UX).

Creating the Oracle switch load file

The simplest method for creating the Oracle switch load file is to use the sample file. The source code used to create the Oracle switch load file is shown in Figure 32.

```
#include <cmqc.h>
#include "xa.h"

extern struct xa_switch_t xaosw;

struct xa_switch_t * MQENTRY MQStart(void)
{
   return(&xaosw);
}
```

Figure 32. Source code for Oracle switch load file, oraswit.c

The xa.h header file that is included is shipped with MQSeries in the same directory as oraswit.c.

Creating the Oracle switch load file on UNIX systems

To create the Oracle switch load file on UNIX systems, oraswit.c must be compiled and linked against libclntsh.

To build the switch load file for:

- AIX, perform steps 1, 2, 3, and 6
- Sun Solaris, perform steps 1, 2, 4, and 6
- HP-UX, perform steps 1, 2, 5, and 6

1. Create the directory into which the Oracle switch load file, oraswit, will be built.
2. Copy the following files from **mqmtop**/samp/xatm into this directory:
   - xa.h
   - oraswit.c
   - xaswit.mak (if using Oracle7)
   - xaswito8.mak (if using Oracle8)
3. Use the source shown in Figure 33 on page 188, which forms part of xaswit.mak on AIX, to build Oracle switch file.

```
ORALIBS=-l clntsh -l m
ORALIBPATH=-L $(ORACLE_HOME)/lib

oraswit:
        xlc_r -e MQStart $(ORALIBPATH) $(ORALIBS) -o $@ oraswit.c
```

Figure 33. Makefile for Oracle7 switch load file on AIX

```
ORALIBS=-l clntsh -l m
ORALIBPATH=-L $(ORACLE_HOME)/lib

ora8swit:
        xlc_r -e MQStart $(ORALIBPATH) $(ORALIBS) -o $@ oraswit.c
```

Figure 34. Makefile for Oracle8 switch load file on AIX

4. Use the source shown in Figure 35 and Figure 36, which form part of xaswit.mak and xaswito8.mak on Sun Solaris, to build the appropriate Oracle switch file on Sun Solaris.

```
ORALIBS=-l clntsh -l m
ORALIBPATH=-L $(ORACLE_HOME)/lib  -R $(ORACLE_HOME)/lib

oraswit:
        $(CC) -G -e MQStart $(ORALIBPATH) $(ORALIBS) -o $@ oraswit.c
```

Figure 35. Makefile for Oracle7 switch load file on Sun Solaris

```
ORALIBS=-l clntsh -l m
include $(ORACLE_HOME)/precomp/lib/env_precomp.mk
PROLDLIBS=$(LLIBCLNTSH) $(SCOREPT) $(SSCOREED)$(DEF_ON) $(LLIBCLIENT) $(LLIBSQL)
          $(STATICTTLIBS)
ORALIBPATH=-R $(ORACLE_HOME)/lib -L $(ORACLE_HOME)/lib$(PROLDLIBS)

ora8swit:
      $(CC) -G -e MQStart $(ORALIBPATH) $(ORALIBS) -o $ @ oraswit.c
```

Figure 36. Makefile for Oracle8 switch load file on Sun Solaris

5. Use the source shown in Figure 37 and Figure 38, which form part of xaswit.mak and xaswito8.mak on HP-UX, to build the appropriate Oracle switch load file on HP-UX.

```
ORALIBS=-l clntsh -l m
ORALIBPATH=+b $(ORACLE_HOME)/lib  -L $(ORACLE_HOME)/lib

oraswit:
        $(CC) -c -Ae +z oraswit.c

        ld -b -e MQStart $(ORALIBPATH) $(ORALIBS) -o oraswit oraswit.o
```

Figure 37. Makefile for Oracle7 switch load file on HP-UX

```
ORALIBS=-l clntsh -l m
ORALIBPATH=+b $(ORACLE_HOME)/lib $(LIBCLNTSH) $(LIBSQL)-L $(ORACLE_HOME)/lib

ora8swit:
       $(CC) -c -Ae +z oraswit.c

       ld -b -e MQStart $(ORALIBPATH) $(ORALIBS) -o oraswit oraswit.o
```

Figure 38. Makefile for Oracle8 switch load file on HP-UX

6. Issue a `make -f xaswit.mak oraswit` command (for Oracle7) or a `make -f xaswito8.mak ora8swit` command (for Oracle8) to build the Oracle switch using the sample makefile.

Adding XAResourceManager configuration information for Oracle

The next step is to modify the qm.ini configuration file of the queue manager, or, for MQSeries for Windows NT only, the Windows NT Registry, to define Oracle as a participant in global units of work. You need to add an XAResourceManager stanza with the following attributes:

Name=name
This attribute is mandatory. Choose a suitable name for this participant. You could include the name of the database being updated.

SwitchFile=name
This attribute is mandatory. The fully-qualified name of the Oracle switch load file

XAOpenString=string

The XA open string for Oracle has the following format:

```
Oracle_XA+Acc=P//|P/userName/passWord
         +SesTm=sessionTimeLimit
         [+DB=dataBaseName]
         [+GPwd=P/groupPassWord]
         [+LogDir=logDir]
         [+MaxCur=maximumOpenCursors]
         [+SqlNet=connectString]
```

where:

Acc= Is mandatory and is used to specify user access information. **P//** indicates that no explicit user or password information is provided and that the **ops$login** form is to be used. **P**/userName/passWord indicates a valid ORACLE user ID and the corresponding password.

SesTm= Is mandatory and is used to specify the maximum amount of time that a transaction can be inactive before the system automatically deletes it. The unit of time is in seconds.

DB= Is used to specify the database name, where DataBaseName is the name Oracle precompilers use to identify the database. This field is required only when applications explicitly specify the database name (that is, use an AT clause in their SQL statements).

GPwd= GPwd is used to specify the server security password, where P/groupPassWord is the server security group password name. Server security groups provide an extra level of protection for different applications running against the same ORACLE instance. The default is an ORACLE-defined server security group.

LogDir= LogDir is used to specify the directory on a local machine where the Oracle XA library error and tracing information can be logged. If a value is not specified, the current directory is assumed. Make sure that user mqm has write-access to this directory.

MaxCur=
MaxCur is used to specify the number of cursors to be allocated when the database is opened. It serves the same purpose as the precompiler option, maxopencursors.

SqlNet= SqlNet is used to specify the SQL*Net connect string that is used to log on to the system. The connect string can be either an SQL*Net V1 string, SQL*Net V2 string, or SQL*Net V2 alias. This field is required when you are setting up Oracle on a machine separate from the queue manager.

See the *Oracle7 Server Distributed Systems, Volume 1: Distributed Data* book (Part Number A32543-1) or the *Oracle8 Server Application Developer's Guide* (Part Number A54642-01) for more information.

XACloseString=string

Oracle does not require an XA close string.

ThreadOfControl=THREAD|PROCESS
You do not need to specify this parameter on UNIX platforms.

For fuller descriptions of each of these attributes, see "The XAResourceManager stanza" on page 140.

In Figure 39, the database to be updated is called MQBankDB. Note that it is recommended to add a `LogDir` to the XA open string so that all error and tracing information is logged to the same place. It is assumed that the Oracle switch load file was copied to the /usr/bin directory after it had been created.

```
XAResourceManager:
  Name=Oracle MQBankDB
  SwitchFile=/usr/bin/oraswit
  XAOpenString=Oracle_XA+Acc=P/scott/tiger+SesTm=35+LogDir=/tmp/ora.log+DB=MQBankDB
```

Figure 39. Sample XAResourceManager entry for Oracle on UNIX platforms

Changing Oracle configuration parameters

The queue manager and user applications use the user ID specified in the XA open string when they connect to Oracle.

- **Database privileges (Oracle7 only)**
  The Oracle user ID specified in the open string must have the privileges to access the V$XATRANS$ view.

  The necessary privilege can be given using the following command, where `userID` is the user ID for which access is being given.

```
grant select on V$XATRANS$ to userID;
```

  See "Security considerations" on page 200 for more information about security.

- **Database privileges (Oracle8 only)**
  The Oracle user ID specified in the open string must have the privileges to access the DBA_PENDING_TRANSACTIONS view.

  The necessary privilege can be given using the following command, where `userID` is the user ID for which access is being given.

```
grant select on DBA_PENDING_TRANSACTIONS to userID;
```

  See "Security considerations" on page 200 for more information about security.

- **Additional database connections (Oracle7 and later)**
  You may need to review your LICENSE_MAX_SESSIONS and PROCESSES settings to take into account the additional connections required by processes belonging to the queue manager. See "Database connections" on page 177 for details about the database connections that the queue manager needs for itself.

Sybase configuration

MQSeries supports:

- Sybase XA Server Version 11.1.1 (with latest support level)
- Sybase OpenClient/C Version 11.1.1 (with latest support level)
- Sybase Adaptive Server Version 11.5
- Sybase Embedded SQL/C Version 11.1.1

when used in conjunction with the Sybase XA-Server component Version 11.1.1 on the following platforms:

- AIX 4.2.0 or later (with Sybase patch SWR 7993 applied)
- Sun Solaris 2.6 or later
- Windows NT 4.0 Service Pack 3, or later

You need to perform the following tasks:

1. Check environment variable settings.
2. Create symbolic lines for Sybase libraries (Sun Solaris only).
3. Enable Sybase XA support.
4. Create the Sybase switch load file.
5. Add an XAResourceManager stanza to the qm.ini file, or to the Windows NT Registry.

Enabling Sybase XA support

Within the Sybase XA configuration file, you need to define a Logical Resource Manager (LRM) for each connection you will make to the Sybase server that is being updated.

Refer to the Sybase publication, *XA-Server Integration Guide for Tuxedo*, (Document ID 35002-01-1111-01), and to the *Release Bulletin XA-Server Version 11.1.1* appropriate for your platform for information about configuring the Sybase server for XA support and modifying the Sybase XA configuration file, `$SYBASE/xa_config`.

An example of the contents of `$SYBASE/xa_config` is shown in Figure 40

```
# The first line must always be a comment

[xa]

  LRM=lrmname
  server=servername
  xaserver=xaserververname
```

Figure 40. Example contents of $SYBASE/xa_config

Creating the Sybase switch load file

The simplest method for creating the Sybase switch load file is to use the sample files installed under the mqmtop/samp/xatm directory. The source code used to create the Sybase switch on UNIX platforms is shown in Figure 41.

```
#include <cmqc.h>
#include "xa.h"

extern struct xa_switch_t sybase_xa_switch;

struct xa_switch_t * MQENTRY MQStart(void)
{
   return(&sybase_xa_switch);
}
```

Figure 41. Source code for sybswit.c on UNIX platforms

The xa.h header file is in the same directory as sybswit.c.

The source for the switch load file can be found in:

- The directory mqmtop/samp/xatm for UNIX platforms
- The directory c:\mqm\tools\c\samples\xatm for MQSeries for Windows NT and MQSeries for OS/2 Warp

Linking the XA switch load file with Sybase libraries

As already described in “Creating the Oracle switch load file on UNIX systems” on page 188, the queue manager processes do not inherit the environment variables (LD_LIBRARY_PATH on Sun Solaris, LIBPATH on AIX, and SHLIB_PATH on HP-UX) from the shell. in which the **strmqm** command was issued. In order for the libraries to be found during start up of the queue manager, you can specify a run-time linking path when the switch load file is built. The sample makefiles in the following sections operate in this way.

Creating the Sybase switch load file on UNIX systems

To create the Sybase switch load file on UNIX systems, sybswit.c must be compiled and linked with the relevant Sybase libraries as shown in Figure 42 and Figure 43.

To build the switch load file:

1. Create the directory into which the Sybase switch load file, sybswit, will be built.
2. Copy the following files into this directory:
   - xa.h
   - sybswit.c
   - xaswit.mak
3. Use the source shown in Figure 42, which forms part of xaswit.mak, to build the Sybase switch file on AIX.

```
SYBLIBS = -lxaserver -lct.so -lcs.so -ltcl.so -lcomn.so -lintl.so -lm
SYBLIBPATH=-L $(SYBASE)/lib

sybswit:
        xlc_r -e MQStart $(SYBLIBPATH) $(SYBLIBS) -o $@ sybswit.c
```

Figure 42. Makefile for Sybase switch on AIX

4. Use the source shown in Figure 43, which forms part of xaswit.mak, to build the Sybase switch file on Sun Solaris.

```
SYBLIBS = -lintl -lxaserver -lct -lcs -lcomn -ltcl -ldl
SYBLIBPATH=-L $(SYBASE)/lib -R $(SYBASE)/lib

sybswit:
        $(CC) -G -e MQStart $(SYBLIBPATH) $(SYBLIBS) -o $@ sybswit.c
```

Figure 43. Makefile for Sybase switch on Sun Solaris

5. Issue a `make -f xaswit.mak sybswit` command to build the Sybase switch using the sample makefile.

Creating the Sybase switch load file on Windows NT

To create the Sybase switch load file on Windows NT, sybswit.c must be compiled and linked with libxaserver.lib.

```
#include <cmqc.h>                                /* MQ header                */
#include "xa.h"                                  /* MQ supplied XA header    */

/*********************************/
/* On NT __STDC__ is not defined */
/*********************************/
struct stdcall_xa_switch_t               /* as defined/used by Sybase on NT   */
{
  char name[RMNAMESZ];         /* name of resource manager          */
  long flags;                            /* resource manager specific options */
  long version;                          /* must be 0                         */

  int (__stdcall * xa_open_entry)();     /* xa_open function pointer          */
  int (__stdcall * xa_close_entry)();    /* xa_close function pointer         */
  int (__stdcall * xa_start_entry)();    /* xa_start function pointer         */
  int (__stdcall * xa_end_entry)();      /* xa_end function pointer           */
  int (__stdcall * xa_rollback_entry)(); /* xa_rollback function pointer      */
  int (__stdcall * xa_prepare_entry)();  /* xa_prepare function pointer       */
  int (__stdcall * xa_commit_entry)();   /* xa_commit function pointer        */
  int (__stdcall * xa_recover_entry)();  /* xa_recover function pointer       */
  int (__stdcall * xa_forget_entry)();   /* xa_forget function pointer        */
  int (__stdcall * xa_complete_entry)(); /* xa_complete function pointer      */

};

/*****************************************************************************/
/* External data declarations                                                */
/*****************************************************************************/
/* specify that the Sybase XA switch uses __stdcall conventions */
extern __declspec( dllimport ) struct stdcall_xa_switch_t sybase_xa_switch;


/* Function Prototypes (called by queue manager) */
int __cdecl intermediate_xa_open_entry(char * a, int b, long c);
int __cdecl intermediate_xa_close_entry(char * a, int b, long c);
int __cdecl intermediate_xa_start_entry(XID *, int, long);
int __cdecl intermediate_xa_end_entry(XID *, int, long);
int __cdecl intermediate_xa_rollback_entry(XID *, int, long);
int __cdecl intermediate_xa_prepare_entry(XID *, int, long);
int __cdecl intermediate_xa_commit_entry(XID *, int, long);
int __cdecl intermediate_xa_recover_entry(XID *, long, int, long);
int __cdecl intermediate_xa_forget_entry(XID *, int, long);
int __cdecl intermediate_xa_complete_entry(int *, int *, int, long);
```

Figure 44 (Part 1 of 3). Source code for sybswit.c on Windows NT

```
/* This intermediate switch is of type declared in xa.h - __cdecl funs */
struct xa_switch_t intermediate_xa_switch =
{
  "SYBASE_XA_SERVER",
  TMNOMIGRATE,
  0,
  intermediate_xa_open_entry,
  intermediate_xa_close_entry,
  intermediate_xa_start_entry,
  intermediate_xa_end_entry,
  intermediate_xa_rollback_entry,
  intermediate_xa_prepare_entry,
  intermediate_xa_commit_entry,
  intermediate_xa_recover_entry,
  intermediate_xa_forget_entry,
  intermediate_xa_complete_entry
};
/*****************************************************************************/
/*                                                                           */
/* Function name:  MQStart                                                   */
/*                                                                           */
/* Description: The queue manager calls this function to access the XA switch */
/*              of Sybase                                                    */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* Input Parameters:  None                                                   */
/*                                                                           */
/* Output Parameters: None                                                   */
/*                                                                           */
/* Returns:           Pointer to Sybase XA switch (now the intermediate swit) */
/*                                                                           */
/*****************************************************************************/
__declspec(dllexport) struct xa_switch_t * MQENTRY MQStart(void)
{
  return( &intermediate_xa_switch );
}

int __cdecl intermediate_xa_open_entry(char * a, int b, long c)
{
  return( sybase_xa_switch.xa_open_entry( a, b, c ) );
}

int __cdecl intermediate_xa_close_entry(char * a, int b, long c)
{
  return( sybase_xa_switch.xa_close_entry( a, b, c ) );
}

int __cdecl intermediate_xa_start_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_start_entry( a, b, c ) );
}

int __cdecl intermediate_xa_end_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_end_entry( a, b, c ) );
}
```

Figure 44 (Part 2 of 3). Source code for sybswit.c on Windows NT

```
int __cdecl intermediate_xa_rollback_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_rollback_entry( a, b, c ) );
}

int __cdecl intermediate_xa_prepare_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_prepare_entry( a, b, c ) );
}

int __cdecl intermediate_xa_commit_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_commit_entry( a, b, c ) );
}

int __cdecl intermediate_xa_recover_entry(XID *a, long b, int c, long d)
{
  return(sybase_xa_switch.xa_recover_entry( a, b, c, d ));
}

int __cdecl intermediate_xa_forget_entry(XID *a, int b, long c)
{
  return( sybase_xa_switch.xa_forget_entry( a, b, c ) );
}

int __cdecl intermediate_xa_complete_entry(int *a, int *b, int c, long d)
{
  return( sybase_xa_switch.xa_complete_entry( a, b, c,d ) );
}

/* End of sybswit.c                                                    */
```

Figure 44 (Part 3 of 3). Source code for sybswit.c on Windows NT

The DEF file shown in Figure 45 is needed to produce the DLL.

```
LIBRARY SYBSWIT

EXPORTS
   MQStart
```

Figure 45. Source code for sybwit.def on Windows NT

To create the DLL:

1. Create a directory into which you want the switch file to be built. The switch file must be defined to MQSeries as a fully-qualified name so the DLL does not need to be built into a directory with LIBPATH. Makefiles are provided for building the switch file using both the Microsoft Visual C++ and IBM VisualAge® for C++ compilers.
2. Copy the following files from \mqm\tools\c\samples\xatm into your new directory:
   - xa.h
   - sybswit.c
   - sybswit.def
3. Copy **either**:
   - xaswit.mak if you are using Microsoft Visual C++
   - xaswiti.mak if you are using IBM VisualAge for C++

4. Use the source code shown in Figure 46 on page 198, which forms part of xaswit.mak on Windows NT, to build the switch load file using the Microsoft Visual C++ compiler.

```
!include <ntwin32.mak>

sybswit.lib sybswit.exp: $*.obj $*.def
                         $(implib) -machine:$(CPU) \
                         -def:$*.def $*.obj

sybswit.dll: $*.obj $*.def $*.exp
             $(link) $(dlllflags)    \
             -base:0x1C000000        \
             $*.exp $*.obj           \
             $(conlibsdll) libxaserver.lib

.c.obj:
    $(cc) $(cflags) $(cvarsdll) $*.c
```

Figure 46. Makefile for Sybase switch on Windows NT using Microsoft Visual C++

5. Issue an `nmake -f xaswit.mak sybswit.dll` command to create the Sybase switch load file, sybswit.dll.
6. To create the switch load file using the IBM VisualAge for C++ compiler, use the source code shown in Figure 47, which forms part of xaswiti.mak on Windows NT, to build the switch load file.

```
.\sybswit.obj: \
    sybswit.c \
    {$(INCLUDE);}xa.h
    @echo " Compile "
    icc.exe /Gm /Ti- /Gd /Ge- /Gf- /Fosybswit.obj /C sybswit.c

.\sybswit.exp: \
    .\sybswit.obj
    @echo " Make exp and lib files "
    ilib.exe /Gi:sybswit .\sybswit.obj

.\sybswit.dll: \
    .\sybswit.exp \
    .\sybswit.obj \
    {$(LIB)}libxaserver.lib
    @echo " Link "
    icc.exe @<<
     /B" /de /pmtype:vio /noe /code:RX /data:RW /dll"
     /B" /def"
     /B" /def:libxaserver"
     /B" /nod:sybswit.lib"
     /Fesybswit.dll
     libxaserver.lib
     .\sybswit.exp
     .\sybswit.obj
```

Figure 47. Makefile for Sybase switch on Windows NT using IBM VisualAge for C++

7. Issue an `nmake -f xaswiti.mak sybswit.dll` command to make the Sybase switch load file.

Adding XAResourceManager configuration information for Sybase

Edit the Windows NT Registry using the MQSeries Services snap-in to define Sybase as a participant in global units of work.

You need to add the following XAResourceManager configuration information:

Name=Name
This attribute is mandatory. Choose a suitable name that this participant will be known as. You could include the name of the database being updated.

SwitchFile=name
This attribute is mandatory The fully-qualified name of the Sybase switch load file.

XAOpenString=string
The XA open string for Sybase must have the following syntax:

```
-Uusername -Ppassword -Nconnection_name -Llogfile -Ttype
```

where:

-U Specifies the user access information; *username* is a valid Sybase user ID.

-P Is the password of the specified user.

-N Is the LRM name corresponding to a connection to the database. It should be defined in the Sybase xa_config file as described in “Enabling Sybase XA support” on page 192.

-L Indicates the path name of the file where Sybase XA-Library error information will be logged. Ensure that the user mqm has write privileges over this file. This field is optional; no error information is logged if it is not specified.

-T Specifies the type of logging used. See the Sybase publication *XA-Server Integration Guide for Tuxedo* for a list of log types.

XACloseString=string
Sybase does not require an XA close string.

ThreadOfControl=THREAD|PROCESS
Sybase is not “threadaware”, so specify PROCESS.

In Figure 48, the *MQBankDB* database is associated with the *lrmname* LRM definition in the Sybase XA configuration file, `$SYBASE/xa_config`. A log file should be included if you want XA function calls to be logged.

```
XAResourceManager:
  Name=Sybase MQBankDB
  SwitchFile=/usr/bin/sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
```

Figure 48. Sample XAResourceManager entry for Sybase on UNIX platforms

Multiple database configurations

If you want to configure the queue manager so that updates to multiple databases can be included within global units of work, then you need to add an XAResourceManager stanza for each of the databases.

If the databases are all managed by the same database manager, each stanza defines a separate database belonging to that database manager. Each stanza should specify the same *SwitchFile*, but the contents of the *XAOpenString* will be different because it specifies the name of the database being updated. For example, the stanzas shown in Figure 49 configure the queue manager with the DB2 databases *MQBankDB* and *MQFeeDB* on UNIX platforms.

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=DB2 MQFeeDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQFeeDB
```

Figure 49. Sample XAResourceManager entries for multiple DB2 databases

If the databases to be updated are managed by different database managers then once again an XAResourceManager stanza needs to be added for each. In this case, each stanza specifies a different *SwitchFile*. For example, if the *MQFeeDB* was managed by Oracle instead of DB2 then the following stanzas could be used:

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=Oracle MQFeeDB
  SwitchFile=/usr/bin/oraswit
  XAOpenString=Oracle_XA+Acc=P/scott/tiger+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

Figure 50. Sample XAResourceManager entries for a DB2 and Oracle database

In principle, there is no limit to the number of database instances that can be configured with a single queue manager.

Security considerations

The following information is provided for guidance only. In all cases you should refer to the documentation provided by the database manager concerned to determine the security implications of running your database under the XA model.

An application process denotes the start of a global unit of work using the MQBEGIN verb. The first MQBEGIN call that an application issues connects to each of the participating databases by calling them at their xa_open entry point. All of the database managers provide a mechanism for supplying a user ID and password in their XAOpenString.

If a user ID is specified in the XAOpenString then it is recommended that one with a minimal set of authorizations be chosen. Consult the documentation of the database manager to determine how the application can gain different privileges. This can often be achieved using EXEC SQL CONNECT or EXEC SQL SET CONNECTION.

Note that on UNIX platforms fastpath applications must run with an effective user ID of mqm while making MQI calls.

Administration tasks

In normal operations only a minimal amount of administration is necessary after you have completed the configuration steps. The administration job is made easier because the queue manager is tolerant of database managers not being available. In particular this means that:

- The queue manager can be started at any time without first starting each of the database managers.
- The queue manager does not need to be stopped and restarted if one of the database managers becomes unavailable.

This allows you to start and stop the queue manager independently from the database managers, and vice versa if the database manager supports it.

Whenever contact is lost between the queue manager and a database manager they need to resynchronize when both become available again.

Resynchronization is the process by which any in-doubt units of work involving that database are completed. In general, this occurs automatically without the need for user intervention. The queue manager asks the database manager for a list of units of work in which it is in doubt. Next it instructs the database manager to either commit or rollback each of these in-doubt units of work.

When the queue manager stops, it needs to resynchronize with each database manager instance during restart. When an individual database manager becomes unavailable, only that database manager need be resynchronized the next time the queue manager notices that the database manager is available again.

The queue manager attempts to regain contact with an unavailable database manager automatically as new global units of work are started. Alternatively, the **rsvmqtrn** command can be used to resolve explicitly all in-doubt units of work.

In-doubt units of work

A database manager may be left with in-doubt units of work if contact with the queue manager is lost after the database manager has been instructed to PREPARE. Until the database manager receives the COMMIT or ROLLBACK outcome from the queue manager, it needs to retain the database locks associated with the updates.

Because these locks prevent other applications from updating, or maybe reading, database records, resynchronization needs to take place as soon as possible.

If for some reason you cannot wait for the queue manager to resynchronize with the database automatically, you could use facilities provided by the database

manager to commit or rollback the database updates manually. This is called making a *heuristic* decision and should be used only as a last resort because of the possibility of compromising data integrity; you may end up committing the database updates when all of the other participants rollback, or vice versa.

It is far better to restart the queue manager, or use the **rsvmqtrn** command when the database has been restarted, to initiate automatic resynchronization.

Using the dspmqtrn command

While a database manager is unavailable it is possible to use the **dspmqtrn** command to check the state of outstanding units of work (UOWs) involving that database.

When a database manager becomes unavailable, before the two-phase commit process is entered, any in-flight UOWs in which it was participating are rolled back. The database manager itself rolls back its in-flight UOWs when it next restarts.

The **dspmqtrn** command displays only those units of work in which one or more participants are in doubt, awaiting the COMMIT or ROLLBACK from the queue manager.

For each of these units of work the state of each of the participants is displayed. If the unit of work did not update the resources of a particular resource manager, it is not displayed.

With respect to an in-doubt unit of work, a resource manager is said to have done one of the following things:

Prepared The resource manager is prepared to commit its updates.

Committed The resource manager has committed its updates.

Rolled-back The resource manager has rolled back its updates.

Participated The resource manager is a participant, but has not prepared, committed, or rolled back its updates.

Note that the queue manager does not remember the individual states of the participants when the queue manager restarts. If the queue manager is restarted, but is unable to contact a database manager, then the in-doubt units of work in which that database manager was participating are not resolved during restart. In this case, the database manager is reported as being in *prepared* state until such time as resynchronization has occurred.

Whenever the **dspmqtrn** command displays an in-doubt UOW, it first lists all the possible resource managers that could be participating. These are allocated a unique identifier, *RMId*, which is used instead of the *Name* of the resource managers when reporting their state with respect to an in-doubt UOW.

Figure 51 on page 203 shows the result of issuing the following command:

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB
AMQ7107: Resource manager 2 is DB2 MQFeeDB

AMQ7056: Transaction number 0,1.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A50600002013746573 74]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Figure 51. Sample dspmqtrn output

The output from Figure 51 shows that there are three resource managers associated with the queue manager. The first is the resource manager `0`, which is the queue manager itself. The other two resource manager instances are the `MQBankDB` and `MQFeeDB` DB2 databases.

The example shows only a single in-doubt unit of work. A message is issued for all three resource managers, which means that updates had been made to the queue manager and both DB2 databases within the unit of work.

The updates made to the queue manager, resource manager **0**, have been *committed*. The updates to the DB2 databases are in *prepared* state, which means that DB2 must have become unavailable before it was called to commit the updates to the *MQBankDB* and *MQFeeDB* databases.

The in-doubt unit of work has an external identifier called an XID. This is the identifier that DB2 associates with the updates.

Using the rsvmqtrn command

The output shown in Figure 51 showed a single in-doubt UOW in which the commit decision had yet to be delivered to both DB2 databases.

In order to complete this unit of work, the queue manager and DB2 need to resynchronize when DB2 next becomes available. The queue manager uses the start of new units of work as an opportunity to attempt to regain contact with DB2. Alternatively, you can instruct the queue manager to resynchronize explicitly using the **rsvmqtrn** command. You should do this soon after DB2 has been restarted so that any database locks associated with the in-doubt unit of work are released as quickly as possible.

This is achieved using the **-a** option which tells the queue manager to resolve all in-doubt units of work. In the following example, DB2 had been restarted so the queue manager was able to resolve the in-doubt unit of work:

```
> rsvmqtrn -mMY_QMGR -a

Any in-doubt transactions have been resolved.
```

Mixed outcomes and errors

Although the queue manager uses a two-phase commit protocol this does not completely remove the possibility of some units of work completing with mixed outcomes. This is where some participants commit their updates, and some back out their updates.

Units of work that complete with a mixed outcome have serious implications because shared resources are no longer in a consistent state.

Mixed outcomes are mainly caused when heuristic decisions are made about units of work instead of allowing the queue manager to resolve in-doubt units of work itself.

Whenever the queue manager detects heuristic damage it produces FFST information and documents the failure in its error logs, with one of two messages:

- If a database manager rolled back instead of committing:

  ```
  AMQ7606 A transaction has been committed but one or more resource
          managers have rolled back.
  ```

- If a database manager committed instead of rolling back:

  ```
  AMQ7607 A transaction has been rolled back but one or more resource
          managers have committed.
  ```

Further messages are issued that identify the databases that are heuristically damaged. It is then your responsibility to perform recovery steps local to the affected databases so that consistency is restored. This is a complicated procedure in which you need first to isolate the update that has been wrongly committed or rolled back, then to undo or redo the database change manually.

Damage occurring due to software errors is less likely. Units of work affected in this way have their transaction number reported by message AMQ7112. The participants may be in an inconsistent state.

```
rsvmqtrn -m MY_QMGR

AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB
AMQ7107: Resource manager 2 is DB2 MQFeeDB

AMQ7112: Transaction number 0,1 has encountered an error.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A5060000201374657374]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has rolled back.
```

Figure 52. Sample dspmqtrn output for a transaction in error

The queue manager does not attempt to recover from such failures until the next queue manager restart. In Figure 52, this would mean that the updates to resource manager **1**, the MQBankDB database, would be left in *prepared* state even if the **rsvmqtrn** was issued to resolve the unit of work.

Changing configuration information

After the queue manager has successfully started to coordinate global units of work you should be wary about making changes to any of the XAResourceManager stanzas in the qm.ini file, or in the Windows NT Registry.

If you do need to change the qm.ini file you can do so at any time, but the changes do not take effect until after the queue manager has been restarted. For example, if you need to alter the XA open string passed to a database manager, you need to restart the queue manager for your change to take effect.

Note that if you remove an XAResourceManager stanza you are effectively removing the ability for the queue manager to contact that database manager.

You should ***never*** change the *Name* attribute in any of your XAResourceManager stanzas. This attribute uniquely identifies that database manager instance to the queue manager. If this unique identifier is changed, the queue manager assumes that the database manager instance has been removed and a completely new instance has been added. The queue manager still associates outstanding units of work with the old *Name*, possibly leaving the database in an in-doubt state.

Removing database manager instances

If you do need to remove a database or database manager from your configuration permanently, you should first ensure that the database is not in doubt. You should perform this check before you restart the queue manager. Most database managers provide commands for listing in-doubt transactions. If there are any in-doubt transactions, first allow the queue manager to resynchronize with the database manager before you remove its XAResourceManager stanza.

If you fail to observe this procedure the queue manager still remembers all in-doubt units of work involving that database. A warning message, AMQ7623, is issued every time the queue manager is restarted. If you are never going to configure this database with the queue manager again you can instruct it to forget about these in-doubt transactions using the -r option of the **rsvmqtrn** command.

There are times when you might need to remove an XAResourceManager stanza temporarily. This is best achieved by commenting out the stanza so that it can be easily reinstated at a later time. You may decide to take this action if you are suffering errors every time the queue manager contacts a particular database or database manager. Temporarily removing the XAResourceManager entry concerned allows the queue manager to start global units of work involving all of the other participants. An example of a commented out *XAResourceManager* stanza follows:

```
# This database has been temporarily removed
#XAResourceManager:
#  Name=DB2 MQBankDB
#  SwitchFile=/usr/bin/db2swit
#  XAOpenString=MQBankDB
```

Figure 53. Commented out XAResourceManager stanza

Users of MQSeries for Windows NT Version 5.1 and later must use the MQSeries Services snap-in to change XAResourceManager configuration information in the Windows NT Registry.

Furthermore, because you have to delete a database manager instance when editing configuration information in the Windows NT Registry as opposed to commenting it out, you must take great care to type in the correct name in the `Name` field when reinstating it.

External syncpoint coordination

A global unit of work may also be coordinated by an external X/Open XA-compliant transaction manager. Here the MQSeries queue manager participates in, but does not coordinate, the unit of work.

The flow of control in a global unit of work coordinated by an external transaction manager is as follows:

1. An application informs the external syncpoint coordinator (for example, CICS) that it wants to start a transaction.
2. The syncpoint coordinator informs known resource managers, such as MQSeries, about the current transaction.
3. The application issues calls to resource managers that are associated with the current transaction. For example, the application could issue MQGET calls to MQSeries.
4. The application issues a commit or back-out request to the external syncpoint coordinator.
5. The syncpoint coordinator completes the transaction by issuing the appropriate calls to each resource manager, typically using two-phase commit protocols.

Table 13 lists the external syncpoint coordinators that can provide a two-phase commit process for transactions in which the MQSeries Version 5 products can participate. Minimum versions and releases are shown; later versions or releases, if any, may be used.

Table 13. XA-compliant external syncpoint coordinators

MQSeries	External syncpoint coordinator
MQSeries for AIX	TXSeries for AIX V4.2 BEA Tuxedo V5.1 or V6.1
MQSeries for HP-UX	TXSeries for HP-UX, V4.2 BEA Tuxedo V5.1 or V6.1
MQSeries for Sun Solaris	TXSeries for Sun Solaris, V4.2 Transarc Encina Monitor V2.5 BEA Tuxedo V5.1 or V6.1
MQSeries for Windows NT	TXSeries for Windows NT, V4.2 BEA TUXEDO V5.1 or V6.1

Note: For MQSeries for OS/2 Warp, and for MQSeries for Windows NT with CICS for Windows NT, a single-phase commit process only is supported. For more information, see “Using CICS” on page 208.

See Chapter 13, “Committing and backing out units of work” in the *MQSeries Application Programming Guide* for information about writing and building transactions to be coordinated by an external syncpoint coordinator.

The remainder of this chapter describes how to enable external units of work.

The MQSeries XA switch structure

Each resource manager participating in an externally coordinated unit of work must provide an XA switch structure. This structure defines both the capabilities of the resource manager and the functions that are to be called by the syncpoint coordinator.

MQSeries provides two versions of this structure:

- *MQRMIXASwitch* for static XA resource management
- *MQRMIXASwitchDynamic* for dynamic XA resource management

In the MQSeries for UNIX systems, these structures are located in the following libraries:

`libmqmxa.a` (nonthreaded)
`libmqmxa_r.a` (threaded)

In MQSeries for Windows NT and MQSeries for OS/2 Warp the structures are located in the following libraries:

`mqmxa.dll` (contains only the *MQRMIXASwitch* version)
`mqmenc.dll` (for use with Encina for Windows NT)
`mqmc4swi.dll` (for use with IBM TXSeries for Windows NT)

Some external syncpoint coordinators (not CICS) require that each resource manager participating in a unit of work supplies its name in the name field of the XA switch structure. The MQSeries resource manager name is “MQSeries XA RMI.”

The way in which the MQSeries XA switch structure is linked to a specific syncpoint coordinator is defined by that coordinator. Information about linking the MQSeries XA switch structure with CICS is provided in “Using CICS” on page 208. For information about linking the MQSeries XA switch structure with other XA-compliant syncpoint coordinators, consult the documentation supplied with those products.

The following considerations apply to the use of MQSeries with all XA-compliant syncpoint coordinators:

- The xa_info structure passed on any xa_open call by the syncpoint coordinator includes the name of an MQSeries queue manager. The name takes the same form as the queue-manager name passed to the MQCONN call. If the name passed on the xa_open call is blank, the default queue manager is used.
- Only one queue manager at a time may participate in a transaction coordinated by an instance of an external syncpoint coordinator: the syncpoint coordinator is effectively connected to the queue manager, and is therefore subject to the rule that only one connection at a time is supported.
- All applications that include calls to an external syncpoint coordinator can connect only to the queue manager that is participating in the transaction managed by the external coordinator (because they are already effectively

connected to that queue manager). However, such applications must issue an MQCONN call to obtain a connection handle, and should issue an MQDISC call before they exit.

- A queue manager whose resource updates are coordinated by an external syncpoint coordinator must be started before the external syncpoint coordinator starts. Similarly, the syncpoint coordinator must be ended before the queue manager is ended.
- If you are using an external syncpoint coordinator that terminates abnormally, you should stop and restart your queue manager ***before*** restarting the syncpoint coordinator to ensure that any messaging operations uncommitted at the time of the failure are properly resolved.

Using CICS

The versions of CICS and IBM TXSeries that are XA-compliant (and use a two-phase commit process) are shown in Table 13 on page 206. The note following the table shows the versions that support only a single-phase commit process.

The CICS two-phase commit process

This process applies to those versions of MQSeries that support an XA-compliant external syncpoint coordinator as shown in Table 13 on page 206.

Requirements of the two-phase commit process

When you use the CICS two-phase commit process with MQSeries, note the following requirements:

- MQSeries and CICS must reside on the same physical machine.
- MQSeries does not support CICS on an MQSeries client.
- You must start the queue manager whose name is specified in the XAD resource definition stanza ***before*** you attempt to start CICS. Failure to do this will prevent you from starting CICS if you have added an XAD resource definition stanza for MQSeries to the CICS region.
- Only one MQSeries queue manager can be accessed at a time from a single CICS region.
- A CICS transaction must issue an MQCONN request before it can access MQSeries resources. The MQCONN call must specify the name of the MQSeries queue manager specified on the XAOpen entry of the XAD resource definition stanza for the CICS region. If this entry is blank, the MQCONN request must specify the default queue manager.
- A CICS transaction that accesses MQSeries resources must issue an MQDISC call from the transaction before returning to CICS. Failure to do this may mean that the CICS application server is still connected, leaving queues open.
- You must ensure that the CICS user ID (cics) is a member of the mqm group, so that the CICS code has the authority to call MQSeries.

  For transactions running in a CICS environment, the queue manager adapts its methods of authorization and determining context as follows:

- The queue manager queries the user ID under which CICS runs the transaction. This is the user ID checked by the Object Authority Manager, and is used for context information.
- In the message context, the application type is MQAT_CICS.
- The application name in the context is copied from the CICS transaction name.

Enabling the CICS two-phase commit process

To enable CICS to use a two-phase commit process to coordinate transactions that include MQI calls, you must add a CICS XAD resource definition stanza entry to the CICS region.

Here is an example of adding an XAD stanza entry for MQSeries for UNIX systems:

```
cicsadd –cxad –r<cics_region> \
      ResourceDescription="MQM XA Product Description" \
      SwitchLoadFile="mqmtop/lib/amqzsc" \
      XAOpen=<queue_manager_name>
```

Here is an example of adding an XAD stanza entry for MQSeries for Windows NT, where <Drive> is the drive where MQM is installed (for example, D:).

```
cicsadd –cxad –r<cics_region> \
     ResourceDescription="MQM XA Product Description" \
     SwitchLoadFile="<Drive>:\mqm\bin\mqmc4swi.dll" \
     XAOpen=<queue_manager_name>
```

For information about using the **cicsadd** command, see the *IBM TXSeries Administration Reference* manual, SC33-1563 or the *IBM TXSeries Version 4.2 CICS Administration Guide* for your platform.

Calls to MQSeries on UNIX systems, and MQSeries for Windows NT can be included in a CICS transaction, and the MQSeries resources will be committed or rolled back as directed by CICS. This support is not available to client applications.

You ***must*** issue an MQCONN from your CICS transaction, in order to access MQSeries resources followed by a corresponding MQDISC on exit.

Enabling CICS user exits

Before you attempt to make use of a CICS user exit, you should read the *IBM TXSeries Version 4.2 CICS Administration Guide* for your platform.

A CICS user exit *point* (normally referred to as a "user exit") is a place in a CICS module at which CICS can transfer control to a program that you have written (a user exit *program*), and at which CICS can resume control when your exit program has finished its work.

One of the user exits supplied with CICS is the “Task termination user exit (UE014015).” This exit can be invoked at normal and abnormal task termination (after any syncpoint has been taken).

MQSeries supplies a CICS task termination exit in source and executable form:

Table 14. CICS task termination exits

MQSeries for...	Source	Executable
Windows NT	amqzscgn.c	mqmc1415.dll
UNIX systems	amqzscgx.c	amqzscg

If you are currently using this exit, you must add the MQSeries calls from the supplied exits to your current exits. Integrate the MQ calls into your existing exits at the appropriate place in the program logic. See the comments in the sample source file for help with this.

If you are not currently using this exit, you will need to add a CICS PD program definition stanza entry to the CICS region.

Here is an example of adding a PD stanza entry for UNIX systems:

```
cicsadd –cpd –r<cics_region> \
      PathName="mqmtop/lib/amqzscg" \
      UserExitNumber=15
```

Here is an example of adding a PD stanza entry for Windows NT:

```
cicsadd –cpd –r<cics_region> \
      PathName="<Drive>:\mqm\dll\mqmc4swi.dll" \
      UserExitNumber=15
```

The CICS single-phase commit process

The information in this section applies to CICS for OS/2 Version 2 which supports a single-phase commit only.

Note the following:

- On a single physical machine, a CICS transaction can access any queue manager, subject to the restriction that any transaction can connect to only one queue manager at a time.
- CICS transactions distributed among multiple physical machines are not supported.
- For transactions running in a CICS environment, the queue manager changes its methods of authorization and determining context as follows:
  - For MQSeries for OS/2 Warp, the user ID remains os2.
  - In the message context, the application type is MQAT_CICS.

  - The application name in the context is copied from the CICS transaction name.

- To use CICS as an external syncpoint coordinator, you must install the MQSeries-supplied code for the appropriate CICS user exits.

Enabling CICS user exits

To enable the CICS single-phase commit process, you need to enable the CICS user exits 15 and 17 (see the information about user exits that customize the operator interface in the *CICS for OS/2 Customization Guide*.

Sample exits, providing the minimum required function, are supplied in the forms shown in Table 15.

Table 15. Sample exits

MQSeries for...	CICS for...	Sample source	Sample executable	Library linking service
OS/2	OS/2, V2.0	amqzsc52.c amqzsc72.c	FAAEXP15.DLL FAAEXP17.DLL	mqmcics.lib
OS/2	OS/2, V3	amqzsc53.c amqzsc73.c	FAAEX315.DLL FAAEX317.DLL	mqmcics3.lib

Using the sample exits

If you are not currently using these CICS exits, then copy the relevant DLLs into a directory from where they can be accessed by CICS at CICS runtime. This can be a directory referenced by your OS/2 LIBPATH setting.

If you are currently using these CICS exits, you must add the MQSeries calls from the supplied samples to your current exits. These MQSeries calls, which are valid only within the context of exits 15 or 17, enable support for CICS and disable the internal MQCMIT and MQBACK calls such that they will return MQRC_ENVIRONMENT_ERROR. Integrate the MQ calls (AMQ*) in your existing exits at the appropriate place in the program logic. See the comments in the sample source code for help with this.

Chapter 15. 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, except that any in-flight transactions are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; nonpersistent messages are lost during the process.

The remainder of this chapter introduces the concepts of recovery and restart in more detail, and tells you how to recover if problems occur. It covers the following topics:

Making sure that messages are not lost (logging)

MQSeries records all significant changes to the data controlled by the queue manager in a log.

This includes the creation and deletion of objects (except channels), 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.

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 that contain the data being recorded. You can define the number and size (as explained in Chapter 11, “Configuring MQSeries” on page 127), or take the system default of 3 files.

In MQSeries for UNIX systems, each of the three files defaults to 4 MB. In MQSeries for OS/2 Warp and Windows NT, each of the three files defaults to 1 MB.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used.

In MQSeries for UNIX systems, if you have not changed the log path, log files are created in the directory:

```
/var/mqm/log/QmName
```

In MQSeries for OS/2 Warp and Windows NT, if you have not changed the log path, log files are created in the directory:

```
C:\MQM\LOG\<QMgrName>
```

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. This default allocation can also be changed, as described in Chapter 11, “Configuring MQSeries” on page 127.

The log control file

The log control file contains the information needed to monitor the use of log files, such as their size and location, the name of the next available file, and so on.

Note: 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 are likely to require modification to meet your requirements. For more information, see “Calculating the size of the log” on page 219.

Types of logging

In MQSeries, the number of files that are required for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities: circular logging and linear logging.

Circular logging

Use circular logging if all you want is restart recovery, using the log to roll back transactions that were in progress when the system stopped.

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are 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 they are no longer required to ensure queue manager data recovery. The mechanism for releasing log files for reuse is described in "Checkpointing – ensuring complete recovery" on page 216. For now, you should know that MQSeries uses secondary log files to extend the log capacity as necessary.

Linear logging

Use linear logging if you want both restart recovery and media or forward recovery (recreating lost or damaged data by replaying the contents of the log).

Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so you can always retrieve any record logged from the time that the queue manager was created.

As disk space is finite, you 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 that are said to be active. Active files contain the log entries required to restart the queue manager. The number of active log files is usually the same as the number of primary log files as defined in the configuration files. (See "Calculating the size of the log" on page 219 for information about defining 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" on page 216 for further information about checkpointing.)

You must decide when inactive log files are no longer required. You can archive them, or you can delete them if they are no longer of interest to your operation. Refer to "Managing logs" on page 220 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.

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 consistent point, 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 (that is, units of work) active at the time of the checkpoint.

Checkpoints are generated automatically by MQSeries. They are taken when the queue manager starts, 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.

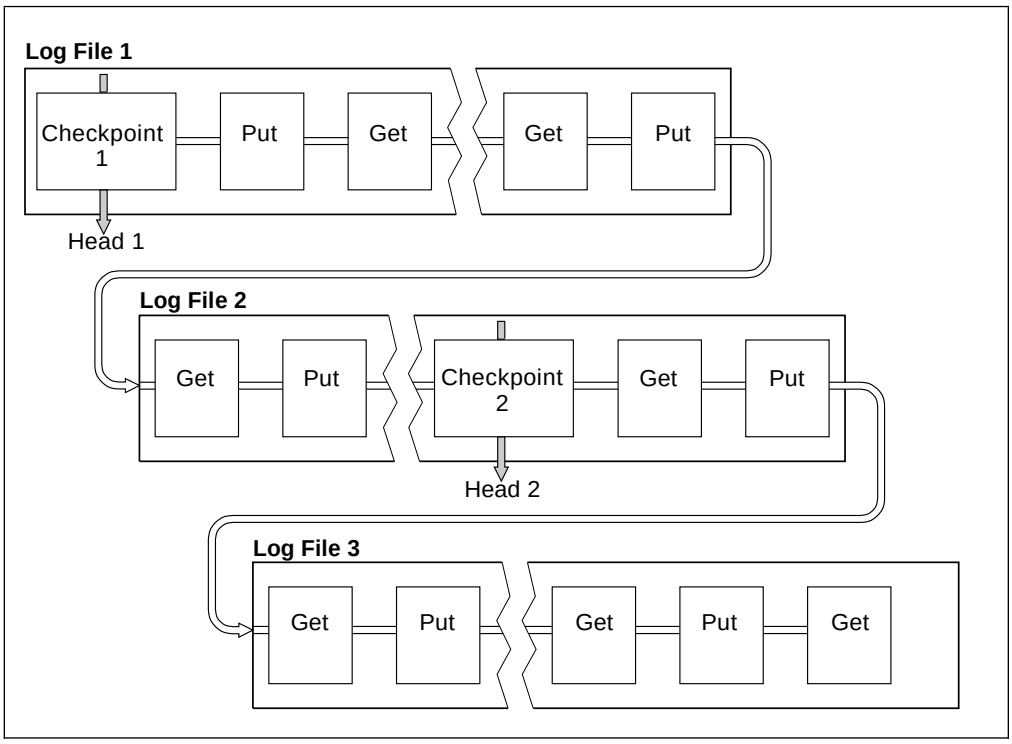


Figure 54. Checkpointing. For simplicity, only the ends of the log files are shown.

In Figure 54, 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.

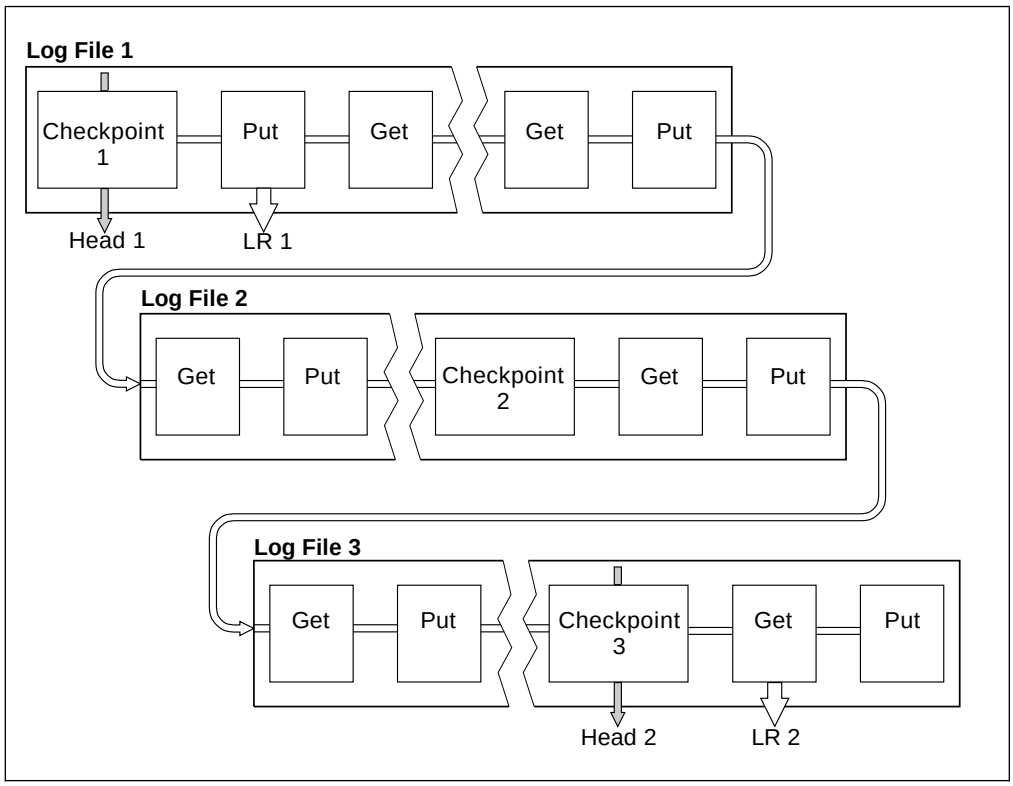


Figure 55. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.

Figure 55 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.

Calculating the size of the log

After deciding whether the queue manager should use circular or linear logging, your next task is to estimate the size of the log that the queue manager will need. The size of the log is determined by the by the following log configuration parameters:

LogFilePages The size of each primary and secondary log file in units of 4 pages
LogPrimaryFiles The number of preallocated primary log files
LogSecondaryFiles The number of secondary log files that can be created for use when the primary log files are full

Table 16 shows the amount of data the queue manager logs for various operations. Most operations performed by the queue manager require a minimal amount of log space, however, when a persistent message is put to a queue, **all** of the message data must be written to the log to make recovery of the message possible. Therefore, the size of the log depends, typically, upon the number and size of the persistent messages the queue manager needs to handle.

Table 16. Log overhead sizes. (All values are approximate.)

Operation	Size
Put persistent message	750 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, 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.

Notes:

1. The number of primary and secondary log files can be changed each time the queue manager is started.
2. The log file size cannot be changed and needs to be determined **before** the queue manager is created.
3. The number of primary log files and the log file size determine the amount of log space that is preallocated when the queue manager is created. You are advised to organize this space as a smaller number of larger log files rather than a larger number of small log files.

4. The total number of primary and secondary log files cannot exceed 63, which, in the presence of long-running transactions, limits the maximum amount of log space that can be made available to the queue manager for restart recovery. The amount of log space the queue manager may need to use for media recovery does not share this limit.
5. When *circular* logging is being used, the queue manager reuses primary log space. This means that the queue manager's log can be smaller than the amount of data you have estimated that the queue manager needs to log. The queue manager will, up to a limit, allocate a secondary log file when a log file becomes full, and the next primary log file in the sequence is not available.
6. Primary log files are made available for reuse during checkpoint. The queue manager takes both the primary and secondary log space into consideration before a checkpoint is taken because the amount of log space is running low.

   If you do not define more primary log files than secondary log files, the queue manager may allocate secondary log files before a checkpoint is taken. This makes the primary log files available for reuse.

Managing logs

Over time, some of the log records written become unnecessary for restarting the queue manager. If you are using circular logging, the queue manager reclaims freed space in the log files. This activity is transparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log 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, message AMQ7463 is issued. In addition, if the log fills because a long-running transaction has prevented the space being released, message AMQ7465 is issued.

Finally, if records are being written to the log faster than the asynchronous housekeeping processes can handle them, message AMQ7466 is issued. If you

see this message, 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 run out of disk space only when it is creating a new file. Therefore, it 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, and you can copy these files to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You 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 “The LogDefaults stanza” on page 132 and “The Log stanza” on page 138.) 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.

In MQSeries for OS/2 Warp, for example, put the log directory on a different drive from that used by the OS/2 swapper file: log files tend to be large, so could fill the disk and prevent expansion of the swapper file.

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 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.

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 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 committed state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when 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 is the re-creation of 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 can be recorded manually, using the **rcdmqimg** command, or automatically in some circumstances.

A media image is a sequence of log records containing an image of an object from which the object itself can be 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 an object is recreated 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 at the following times:

- Images of all process objects and queues that are not local are taken at each shutdown.
- Images of empty local queues are taken at each shutdown.

Media images can also be recorded manually using the **rcdmqimg** command, described in "rcdmqimg (Record media image)" on page 310. Issuing this command causes a media image of the MQSeries object to be written. Once this has been done, only the logs that hold the media image, and all the logs created after this time, are needed to recreate damaged objects. The benefit of doing this depends on such factors as the amount of free storage available, and the speed at which log files are created.

Recovering media images

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 start up. 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 start up operation.

You must recover other objects manually, using the **rcrmqobj** command.

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 can 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)” on page 312 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 start up

If the queue manager discovers a damaged object during start up, 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 is 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 start up processing on a queue manager that does not support media recovery, the queues are marked as damaged objects and the uncommitted messages on them are ignored. This is because it is not possible to perform media recovery of damaged objects on such a queue manager and the only action left is to delete them. Message AMQ7472 is issued to report any damage.

Recovering damaged objects at other times

Media recovery of objects is automatic only during start up. At other times, when object damage is detected, operator message AMQ7472 is issued and most operations using the object fail. If the queue manager object is damaged at any time after the queue manager has started, the queue manager performs a preemptive shutdown. When an object has been damaged you 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)” on page 312 for further details).

Protecting MQSeries log files

It is important that when an MQSeries queue manager is running you do not remove the log files manually. If a user inadvertently (or maliciously) deletes the log files which a queue manager needs to restart, MQSeries **does not** issue any errors and continues to process data *including persistent messages*. The queue manager shuts down normally, but will fail to restart. Media recovery of messages then becomes impossible.

Any user with the authority to remove logs that are being used by an active queue manager also has authority to delete other important queue manager resources (such as authorization files, queue files, the object catalog, and MQSeries executables). They therefore can damage, perhaps through inexperience or even intent, a running or dormant queue manager in a way against which MQSeries cannot protect itself.

Exercise caution when conferring super user or mqm authority.

Backing up and restoring MQSeries

Periodically, you may want 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:

1. Ensure that the queue manager is not running.

   If your queue manager is running, stop it with the **endmqm** command.

   Note: If you try to take a backup of a running queue manager, the backup may not be consistent due to updates in progress when the files were copied.

2. 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, see Chapter 11, “Configuring MQSeries” on page 127.

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

more information about name transformations, see “Understanding MQSeries file names” on page 27.

3. Take copies of all the queue manager’s data and log file directories, including all subdirectories.

   Make sure that you do not miss any 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.

4. Ensure that you preserve the ownerships of the files. For MQSeries for UNIX systems, you can do this with the **tar** command.

Restoring MQSeries

To restore a backup of a queue manager’s data, you must:

1. Ensure that the queue manager is not running.
2. Locate the directories under which the queue manager places its data and its log files. This information is held in the configuration file.
3. Clear out the directories into which you are going to place the backed up data.
4. Copy the backed up queue manager data and log files into the correct places.

Check the resulting directory structure to ensure that you have all of the required directories.

See Appendix B, “Directory structure (UNIX systems)” on page 347 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

This section looks at a number of possible problems and indicates how to recover from them.

Disk drive failures

You may suffer problems with a disk drive containing either the queue manager data, the log, or both. Problems can include data loss or corruption. The three cases differ only in the part of the data that survives, if any.

In ***all*** cases 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 the queue manager does not restart, restore a backup. If the queue manager restarts, check whether any other objects have been damaged using MQSC commands, such as DISPLAY QUEUE. Recover those you find, using the **rcrmqobj** command. For example:

```
rcrmqobj -m QMgrName -t all *
```

  where QMgrName is the queue manager being recovered. -t all * indicates that all objects of any type (except channels) are to be recovered. If only one or two objects have been reported as damaged, you may want to specify those objects by name and type here.
- **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 that 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 preemptive shutdown. There are two ways of recovering in these circumstances depending on the type of logging you use:

- **For linear logging only**, manually delete the file containing the damaged object and restart the queue manager. (You can use the **dspmqfls** command

to determine the real, file-system name of the damaged object.) Media recovery of the damaged object is automatic.

- **For circular or linear logging**, restore the last backup of the queue manager data and log and restart the queue manager.

Damaged single object

If a single object is reported as damaged during normal operation, there are two ways of recovering, depending on the type of logging you use:

- **For linear logging**, recreate the object from its media image.
- **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.

Dumping the contents of the log using the dmpmqlog command

The **dmpmqlog** command can be used to dump the contents of the queue manager log. By default all active log records are dumped, that is, the command starts dumping from the head of the log. Normally this is from the start of the last completed checkpoint.

The log can be dumped only when the queue manager is not running. Because the queue manager takes a checkpoint during shutdown, the active portion of the log usually contains a small number of log records. However, the **dmpmqlog** command can be instructed to dump more log records using one of the following options to change the start position of the dump:

- The simplest option is to start dumping from the *base* of the log. The base of the log is the first log record in the log file that contains the head of the log. The amount of additional data dumped in this case depends upon where the head of the log is positioned in the log file. If it is near to the start of the log file only a small amount of additional data is dumped. If the head is near to the end of the log file then significantly more data is dumped.
- Another option enables the start position of the dump to be specified as an individual log record. Each log record is identified by a unique *log sequence number (LSN)*. In the case of circular logging, this starting log record cannot be prior to the base of the log; this restriction does not apply to linear logs. Inactive log files may need to be reinstated before running the command. For this option a valid LSN must be specified as the start position. This must be taken from previous **dmpmqlog** output.

  For example, with linear logging you could specify the `nextlsn` from your last **dmpmqlog** output. The Next LSN appears in `Log File Header` and indicates the LSN of the next log record to be written. This can therefore be used as a start position to format all log records that have been written since the last time the log was dumped.
- The third option is for linear logs only. The dumper can be instructed to start formatting log records from any given log file extent. In this case the log

dumper expects to find this log file, and each successive one, in the same directory as the active log files. This option does not apply to circular logs, because in this case the log dumper cannot access log records prior to the base of the log.

The output from the **dmpmqlog** command is the `Log File Header` and a series of formatted log records. The queue manager uses several log records to record changes to its data.

Some of the information that is formatted is of use only internally. The following list includes the most useful log records:

Log File Header

Each log has a single log file header, which is always the first thing formatted by the **dmpmqlog** command. It contains the following fields:

logactive The number of primary log extents.

loginactive The number of secondary log extents.

logsize The number of 4 KB pages per extent.

baselsn The first LSN in the log extent containing the head of the log.

nextlsn The LSN of next log record to be written.

headlsn The LSN of the log record at the head of the log.

tailsn The LSN identifying the tail position of the log.

hflag1 Identifies whether log is CIRCULAR or LOG RETAIN (linear).

HeadExtentID The log extent containing the head of the log.

Log Record Header

Each log record within the log has a fixed header containing the following information:

LSN The log sequence number.

LogRecdType The type of the log record.

XTranid The transaction identifier associated with this log record (if any).

A *TranType* of MQI indicates an MQ-only transaction. A *TranType* of XA is involved with other resource managers. Updates involved within the same unit of work have the same *XTranid*.

QueueName The queue associated with this log record (if any).

Qid The unique internal identifier for the queue.

PrevLSN LSN of previous log record within the same transaction (if any).

Start Queue Manager

This logs that the queue manager has been started.

StartDate The date that the queue manager was started.

StartTime The time that the queue manager was started.

Stop Queue Manager
This logs that the queue manager has been stopped.

StopDate The date that the queue manager was stopped.

StopTime The time that the queue manager was stopped.

ForceFlag The type of shutdown that was used.

Start Checkpoint
This denotes the start of a queue manager checkpoint.

End Checkpoint
This denotes the end of a queue manager checkpoint.

ChkPtLSN The LSN of the log record that started this checkpoint.

Put Message
This logs a persistent message put to a queue. If the message was put under syncpoint, then the log record header contains a nonnull *XTranid*. The remainder of the record contains:

SpcIndex An identifier for the message on the queue. It can be used to match the corresponding MQGET that was used to get this message from the queue. In this case a subsequent *Get Message* log record can be found containing the same QueueName and SpcIndex. At this point the SpcIndex identifier can be reused for a subsequent put message to that queue.

Data Contained in the hex dump for this log record is various internal data followed by the Message Descriptor (eyecatcher MD) and the message data itself.

Put Part
Persistent messages that are too large for a single log record are logged as a single *Put Message* record followed by multiple *Put Part* log records.

Data Continues the message data where the previous log record left off.

Get Message
Only gets of persistent messages are logged. If the message was got under syncpoint then the log record header contains a nonnull *XTranid*. The remainder of the record contains:

SpcIndex Identifies the message that was got from the queue. The most recent *Put Message* log record containing the same *QueueName* and *SpcIndex* identifies the message that was got.

QPriority The priority of the message got from the queue.

Start Transaction
Indicates the start of a new transaction. A TranType of MQI indicates an MQ-only transaction. A TranType of XA indicates one that involves other resource managers. All updates made by this transaction will have the same *XTranid*.

Prepare Transaction
Indicates that the queue manager is prepared to commit the updates associated with the specified *XTranid*. This log record is written as part of a two-phase commit involving other resource managers.

Commit Transaction
Indicates that the queue manager has committed all updates made by a transaction.

Rollback Transaction
This log record denotes the queue manager's intention to roll back a transaction.

End Transaction
This log record denotes the end of a rolled-back transaction.

Transaction Table
This record is written during syncpoint. It records the state of each transaction that has made persistent updates. For each transaction the following information is recorded:

XTranid	Transaction identifier.
FirstLSN	LSN of first log record associated with transaction.
LastLSN	LSN of last log record associated with transaction.

Transaction Participants
This log record is written by the XA Transaction Manager component of the queue manager. It records the external resource managers that are participating in transactions. For each participant the following is recorded:

RMName	The name of the resource manager.
RMId	Resource manager identifier. This is also logged in subsequent *Transaction Prepared* log records which record global transactions in which the resource manager is participating.
SwitchFile	The switch load file for this resource manager.
XAOpenString	The XA open string for this resource manager.
XACloseString	The XA open string for this resource manager.

Transaction Prepared
This log record is written by the XA Transaction Manager component of the queue manager. It indicates that the specified global transaction has been successfully prepared. Each of the participating resource managers will be instructed to commit. The *RMId* of each prepared resource manager is recorded in the log record. If the queue manager itself is participating in the transaction a *Participant Entry* with an *RMID* of zero will be present.

Transaction Forget
This log record is written by the XA Transaction Manager component of the queue manager. It follows the *Transaction Prepared* log record when the commit decision has been delivered to each participant.

Purge Queue
This logs the fact that all messages on a queue have been purged, for example, using the RUNMQSC CLEAR command.

Queue Attributes
This logs the initialization or change of the attributes of a queue

Create Object
Logs the creation of an MQSeries object

ObjName The name of the object that was created.

UserId The user ID performing the creation.

Delete Object
Logs the deletion of an MQSeries object

ObjName The name of the object that was deleted.

Figure 56 on page 233 shows example output from a **dmpmqlog** command. The dump, which started at the LSN of a specific log record, was produced using the following command:

```
dmpmqlog -mtestqm -s0:0:0:44162
```

```
AMQ7701: DMPMQLOG command is starting.
LOG FILE HEADER
***************

counter1  . . . : 23                  counter2  . . . : 23
FormatVersion . : 2                   logtype . . . . : 10
logactive . . . : 3                   loginactive . . : 2
logsize . . . . : 1024       pages
baselsn . . . . : <0:0:0:0>
nextlsn . . . . : <0:0:0:60864>
lowtranlsn  . . : <0:0:0:0>
minbufflsn  . . : <0:0:0:58120>
headlsn . . . . : <0:0:0:58120>
taillsn . . . . : <0:0:0:60863>
logfilepath . . : ""
hflag1  . . . . : 1
               -> CONSISTENT
               -> CIRCULAR
HeadExtentID  . : 1                   LastEID . . . . : 846249092
LogId . . . . . : 846249061           LastCommit  . . : 0
FirstArchNum  . : 4294967295          LastArchNum . . : 4294967295
nextArcFile . . : 4294967295          firstRecFile  . : 4294967295
firstDlteFile . : 4294967295          lastDeleteFile  : 4294967295
RecHeadFile . . : 4294967295          FileCount . . . : 3
frec_trunclsn . : <0:0:0:0>
frec_readlsn  . : <0:0:0:0>
frec_extnum . . : 0                   LastCId . . . . : 0
onlineBkupEnd . : 0                   softmax . . . . : 4194304


LOG RECORD - LSN <0:0:0:44162>
**********

HLG Header: lrecsize 212, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Start Checkpoint (1025)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 192                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

No data for Start Checkpoint Record
```

Figure 56 (Part 1 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:44374>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Transaction Table (773)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
TranCount . . . : 0

LOG RECORD - LSN <0:0:0:44594>
**********

HLG Header: lrecsize 1836, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Participants (1537)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 1816                LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPH
Version . . . . : 1                   Flags . . . . . : 3
Count . . . . . : 2

Participant Entry 0
RMName  . . . . : DB2 MQBankDB
RMId  . . . . . : 1
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :

Participant Entry 1
RMName  . . . . : DB2 MQBankDB
RMId  . . . . . : 2
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :
```

Figure 56 (Part 2 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:46448>
**********

HLG Header: lrecsize 236, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM End Checkpoint (1026)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 216                  LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

ChkPtLSN  . . . : <0:0:0:44162>
OldestLSN . . . : <0:0:0:0>
MediaLSN  . . . : <0:0:0:0>


LOG RECORD - LSN <0:0:0:52262>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 200                  LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000
```

Figure 56 (Part 3 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:52482>
**********

HLG Header: lrecsize 730, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 710                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : Queue1
Qid . . . . . . : {Hash 196836031, Counter: 0}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:52262>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                Data.Length . . : 486
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0       ..............¿
00032:  00 00 00 00 00 00 00 01 00 00 00 22 00 00 00 00    ...........".....
00048:  00 00 00 00 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3ä-.........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 22 00 00 00 00 00 00 00 00    .......".........
00144:  00 00 00 00 00 00 00 C9 2C B5 C0 25 FF FF FF FF    .......,[¿%....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74                test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C                sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 32 31 35 32 30 20 20 20 20 00 00 00 00    10421520    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  50 65 72 73 69 73 74 65 6E 74 20 6D 65 73 73 61    Persistent messa
00464:  67 65 20 70 75 74 20 75 6E 64 65 72 20 73 79 6E    ge put under syn
00480:  63 70 6F 69 6E 74                                  cpoint
```

Figure 56 (Part 4 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:53458>
**********

HLG Header: lrecsize 734, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 714                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: NULL
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                Data.Length . . : 490
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0       ..............¿
00032:  00 00 00 00 00 00 00 01 00 00 00 26 00 00 00 00    ...........&....
00048:  00 00 00 00 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3ä-........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 26 00 00 00 00 00 00 00 00    .......&........
00144:  00 00 00 00 00 00 00 C9 2C B6 D8 DD FF FF FF FF    .......,.”.....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74                test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C                sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 33 32 37 30 36 20 20 20 20 00 00 00 00    10432706    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  50 65 72 73 69 73 74 65 6E 74 20 6D 65 73 73 61    Persistent messa
00464:  67 65 20 6E 6F 74 20 70 75 74 20 75 6E 64 65 72    ge not put under
00480:  20 73 79 6E 63 70 6F 69 6E 74                       syncpoint
```

Figure 56 (Part 5 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:54192>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 1}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:52482>

Version . . . . : 1
LOG RECORD - LSN <0:0:0:54408>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000

LOG RECORD - LSN <0:0:0:54628>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Get Message (259)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 220                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : Queue1
Qid . . . . . . : {Hash 196836031, Counter: 0}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:54408>

Version . . . . : 2
SpcIndex  . . . : 1                   QPriority . . . : 0
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 4294967295, Low 4294967295}
```

Figure 56 (Part 6 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:54868>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Get Message (259)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 220                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: NULL
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 2
SpcIndex  . . . : 1                   QPriority . . . : 0
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 4294967295, Low 4294967295}
LOG RECORD - LSN <0:0:0:55108>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: MQI    TranNum{High 0, Low 3}
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:54628>

Version . . . . : 1

LOG RECORD - LSN <0:0:0:55324>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Start Transaction (769)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737 4716D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
SoftLogLimit  . : 10000
```

Figure 56 (Part 7 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:55544>
**********

HLG Header: lrecsize 738, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : AQM Put Message (257)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 718                 LogRecdOwnr . . : 256    (AQM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : Queue2
Qid . . . . . . : {Hash 184842943, Counter: 2}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:55324>

Version . . . . : 3
SpcIndex  . . . : 1
PrevLink.Locn . : 36                  PrevLink.Length : 8
PrevDataLink  . : {High 0, Low 2048}
Data.Locn . . . : 2048                Data.Length . . : 494
Data  . . . . . :
00000:  41 51 52 48 00 00 00 04 FF FF FF FF FF FF FF FF    AQRH............
00016:  00 00 00 00 00 00 00 00 00 00 00 01 00 01 01 C0    ...............¿
00032:  00 00 00 00 00 00 00 01 00 00 00 2A 00 00 00 00    ...........*....
00048:  00 00 00 01 41 4D 51 20 74 65 73 74 71 6D 20 20    ....AMQ testqm
00064:  20 20 20 20 33 80 2D D2 00 00 10 13 00 00 00 00        3ä-........
00080:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00096:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00112:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01    ................
00128:  00 00 00 00 00 00 00 2A 00 00 00 00 00 00 00 00    .......*........
00144:  00 00 00 00 00 00 00 C9 2C B8 3E E8 FF FF FF FF    .......,fl>.....
00160:  4D 44 20 20 00 00 00 01 00 00 00 00 00 00 00 08    MD  ............
00176:  00 00 00 00 00 00 01 11 00 00 03 33 20 20 20 20    ...........3
00192:  20 20 20 20 00 00 00 00 00 00 00 01 20 20 20 20        ........
00208:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00224:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00240:  20 20 20 20 20 20 20 20 20 20 20 74 65 73 74                test
00256:  71 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20    qm
00272:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00288:  20 20 20 20 20 20 20 20 20 20 20 73 62 6F 6C                sbol
00304:  61 6D 20 20 20 20 20 20 04 37 34 38 30 00 00 00    am      .7480...
00320:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00336:  00 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20    ........
00352:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00368:  20 20 20 20 20 20 20 20 00 00 00 06 75 74 7A 61            ....utza
00384:  70 69 20 20 20 20 20 20 20 20 20 20 20 20 20 20    pi
00400:  20 20 20 20 20 20 20 20 31 39 39 37 30 35 31 39            19970519
00416:  31 30 34 34 35 38 37 32 20 20 20 20 00 00 00 00    10445872    ....
00432:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00448:  41 6E 6F 74 68 65 72 20 70 65 72 73 69 73 74 65    Another persiste
00464:  6E 74 20 6D 65 73 73 61 67 65 20 70 75 74 20 75    nt message put u
00480:  6E 64 65 72 20 73 79 6E 63 70 6F 69 6E 74          nder syncpoint
```

Figure 56 (Part 8 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:56282>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Prepare Transaction (770)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:55544>

Version . . . . : 1


LOG RECORD - LSN <0:0:0:56498>
**********

HLG Header: lrecsize 708, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Prepared (1538)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 688                 LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPR
Version . . . . : 1                   Flags . . . . . : 1
Count . . . . . : 3

Participant Entry 0
RMId  . . . . . : 0                   State . . . . . : 2

Participant Entry 1
RMId  . . . . . : 1                   State . . . . . : 2

Participant Entry 2
RMId  . . . . . : 2                   State . . . . . : 2
```

Figure 56 (Part 9 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:57206>
**********

HLG Header: lrecsize 216, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Commit Transaction (774)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 196                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:56282>

Version . . . . : 1
LOG RECORD - LSN <0:0:0:57440>
**********

HLG Header: lrecsize 224, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Forget (1539)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 204                 LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: XA
   XID: formatID 5067085, gtrid_length 14, bqual_length 4
        gtrid [3270BDB4000010237465737471 6D]
        bqual [00000001]
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLFG
Version . . . . : 1                   Flags . . . . . : 0
```

Figure 56 (Part 10 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:58120>
**********

HLG Header: lrecsize 212, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Start Checkpoint (1025)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 192                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

No data for Start Checkpoint Record

LOG RECORD - LSN <0:0:0:58332>
**********

HLG Header: lrecsize 220, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ATM Transaction Table (773)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 200                 LogRecdOwnr . . : 768    (ATM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
TranCount . . . : 0
```

Figure 56 (Part 11 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:58552>
**********

HLG Header: lrecsize 1836, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : Transaction Participants (1537)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 1816                LogRecdOwnr . . : 1536   (T)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Id. . . . . . . : TLPH
Version . . . . : 1                   Flags . . . . . : 3
Count . . . . . : 2

Participant Entry 0
RMName  . . . . : DB2 MQBankDB
RMId  . . . . . : 1
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :

Participant Entry 1
RMName  . . . . : DB2 MQFeeDB
RMId  . . . . . : 2
SwitchFile  . . : /Development/sbolam/build/devlib/tstxasw
XAOpenString  . :
XACloseString . :

LOG RECORD - LSN <0:0:0:60388>
**********

HLG Header: lrecsize 236, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM End Checkpoint (1026)
Eyecatcher  . . : ALRH                Version . . . . : 1
LogRecdLen  . . : 216                 LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

ChkPtLSN  . . . : <0:0:0:58120>
OldestLSN . . . : <0:0:0:0>
MediaLSN  . . . : <0:0:0:0>
```

Figure 56 (Part 12 of 13). Example dmpmqlog output

```
LOG RECORD - LSN <0:0:0:60624>
**********

HLG Header: lrecsize 240, version 1, rmid 0, eyecatcher HLRH

LogRecdType . . : ALM Stop Queue Manager (1028)
Eyecatcher  . . : ALRH                 Version . . . . : 1
LogRecdLen  . . : 220                  LogRecdOwnr . . : 1024   (ALM)
XTranid . . . . : TranType: NULL
QueueName . . . : NULL
Qid . . . . . . : {NULL_QID}
ThisLSN . . . . : <0:0:0:0>
PrevLSN . . . . : <0:0:0:0>

Version . . . . : 1
StopDate  . . . : 19970519             StopTime  . . . : 10490868
SessionNumber . : 0                    ForceFlag . . . : Quiesce

AMQ7702: DMPMQLOG command has finished successfully.
```

Figure 56 (Part 13 of 13). Example dmpmqlog output

Notes for Figure 56 on page 233:

1. The *headlsn* in the *Log File Header* has a value of <0:0:0:58120>. This is where the dump would have started had we not requested a different starting LSN.
2. The *nextlsn* is <0:0:0:60864> which will be the LSN of the first log record that the queue manager will write when it is next restarted.
3. The *HeadExtentID* is 1, indicating that the head of the log currently resides in log file S0000001.LOG.
4. The first log record formatted is a *Start Checkpoint* log record. The checkpoint spans a number of log records until the *End CheckPoint* record at <0:0:0:46448>.
5. One of the records logged during checkpoint is the *Transaction Participants* log record at <0:0:0:44594>. This details the resource managers that participate in global transactions coordinated by the queue manager.
6. The *Start Transaction* log record at <0:0:0:52262> denotes the start of a transaction. The *XTranid* shows a *TranType* of MQI, which indicates that it is a local transaction including MQ updates only.
7. The next log record is a *Put Message* log record that records the persistent MQPUT under the syncpoint that started the transaction. The MQPUT was made to the queue *Queue1* and the message data is logged as `Persistent message put under syncpoint`. This message has been allocated a *SpcIndex* of 1, which will be matched to the later MQGET of this message.
8. The next log record at LSN <0:0:0:53458> is also a *Put Message* record. This persistent message was put to a different queue, *Queue2*, but was not made under syncpoint since the *XTranid* is *NULL*. It too has a *SpcIndex* of 1, which is a unique identifier for this particular queue.
9. The next log record at LSN <0:0:0:54192> commits the message that was put under syncpoint.

10. In log records <0:0:0:54408> and <0:0:0:54628> a new transaction is started by an MQGET under syncpoint for queue *Queue1*. The *SpcIndex* in the *Get Message* log record is 1 indicating that this was the same message that was put to *Queue1* in <0:0:0:52262>.
11. The next log record gets the message that was put to *Queue2* by the other *Put Message* log record.
12. The MQGET under syncpoint has been committed as indicated by the *Commit Transaction* log record at <0:0:0:55108>.
13. Finally an MQBEGIN is used to start a global transaction in the *Start Transaction* log record at <0:0:0:55324>. The *XTranid* in this log record has a *TranType* of XA.
14. The following *Put Message* records a persistent message put to *Queue2*. This shares the same *XTranid* as the previous log record.
15. If a *Transaction Prepared* log record is written for this *Xtranid* then the transaction as a whole must be committed. The absence of such a log record can be taken as an indication that the transaction was rolled back. In this case a *Transaction Prepared* log record is found at <0:0:0:56498>. This records the queue manager itself as a participant with an *RMId* of zero. There are two further participants, their *RMIds* of 1 and 2 can be matched with the previous *Transaction Participants* log record.
16. During the commit phase the XA Transaction Manager component of the queue manager does not log individual responses from the participants. The log indicates only whether the queue manager updates were committed or not. The *Commit Transaction* log record at <0:0:0:57206> indicates that the message was indeed committed to *Queue2*.
17. The *Transaction Forget* log record at <0:0:0:57440> indicates that the commit decision was also delivered to the other two resource managers. Any failure of these resource managers to commit their updates will have been diagnosed in the queue manager's error logs.

Chapter 16. Problem determination

This chapter suggests reasons for some of the problems you may have using MQSeries. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

Problem determination is not problem solving. However, the process of problem determination often enables you to solve a problem. For example, if you find that the cause of the problem is an error in an application program, you can solve the problem by correcting the error.

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 IBM Support Center. This chapter contains these sections:

Preliminary checks

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

The cause of your problem could be in:

- MQSeries
- The network
- The application

The sections that follow raise some fundamental questions that you need to consider. As you work through the questions, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause immediately, they could be useful later if you have to carry out a systematic problem determination exercise.

Has MQSeries run successfully before?

If MQSeries has not run successfully before, it is likely that you have not yet set it up correctly. See one of the following publications to check that you have installed the product correctly, and ensure that the Installation Verification Test (IVT) has been run:

- "Chapter 3. Installing the MQSeries for AIX Server" in the *MQSeries for AIX V5.1 Quick Beginnings* book
- "Chapter 3. Installing MQSeries for OS/2 Warp" in the *MQSeries for OS/2 Warp V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for HP-UX Server" in the *MQSeries for HP-UX V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for Sun Solaris Server" in the *MQSeries for Sun Solaris V5.1 Quick Beginnings* book
- Chapter 4, "Installing MQSeries for Windows NT" in the *MQSeries for Windows NT V5.1 Quick Beginnings* book

Also look at the *MQSeries Intercommunication* book for information about post-installation configuration of MQSeries.

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" on page 259 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 Chapter 5, "Return codes" in the *MQSeries Application Programming Reference* manual for a description of that return code.

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which it is reproduced:

- Is it caused by a command or an equivalent administration request?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.

- Is it caused by a program? Does it fail on all 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?
- Have you changed any component of the operating system that could affect the operation of MQSeries? For example, have you modified the Windows NT Registry hive?

Has the application run successfully before?

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Before you answer **Yes** to this question, consider the following:

- Have any changes been made to the application since it last ran successfully?

  If so, it is likely that the error lies somewhere in the new or modified part of the application. Take a look at the changes and see if you can find an obvious reason for the problem. Is it possible to retry using a back level of the application?

- Have all the functions of the application been fully exercised before?

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

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

- Does the application check all return codes?

  Has your MQSeries system been changed, perhaps in a minor way, such that your application does not check the return codes it receives as a result of the change. For example, does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?

- Does the application run on other 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 Guide* 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.

Common programming errors

The errors in the following list 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.
- Failing to initialize *Encoding* and *CodedCharSetId* following MQRC_TRUNCATED_MSG_ACCEPTED.

Problems with commands

You should be careful when including special characters, for example, back slash (\) and double quote (”) characters, in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a \, that is, enter \\ or \” if you want \ or ” in your text.

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, or changed any MQSeries definitions, that might account for the problem?

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so these are the times when load-dependent problems are most likely to occur. (If your 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 IBM Support Center for any PTF error.
- If a PTF has been applied to any other program, consider the effect it might have on the way MQSeries interfaces with it.

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” on page xvi) 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 incorrect output?” on page 252
- “Have you failed to receive a response from a PCF command?”
- “Does the problem affect only remote queues?” on page 254
- “Is your application or system running slowly?” on page 254

If none of these symptoms describe your problem, consider whether it might have been caused by another component of your system.

Have you obtained incorrect output?

In this book, “incorrect output” refers to your application:

- Not receiving a message that it was expecting.
- Receiving a message containing unexpected or corrupted information.
- Receiving a message that it was not expecting, for example, one that was destined for a different application.

In all cases, check that any queue or queue manager aliases that your applications are using are correctly specified and accommodate any changes that have been made to your network.

If an MQSeries error message is generated, all of which are prefixed with the letters “AMQ,” you should look in the error log. See “Error logs” on page 259 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 “MQDLH - Dead-letter header” in the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure (MQDLH).

  If the dead-letter queue contains messages, you can use the provided browse sample application (amqsbcg) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

- Has a message been sent to the error log?

  See “Error logs” on page 259 for further information.

- Are the queues enabled for put and get operations?

- Is the *WaitInterval* long enough?

  If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See “MQGMO - Get-message options” and “MQGET - Get message” in 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 reveal the problem, try stopping and restarting MQSeries, responding to any messages that are produced in the error log.

If the problem still occurs after restart, contact your IBM Support Center for help.

Are some of your queues failing?

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems:

1. Display the information about each queue. You can use the MQSC command DISPLAY QUEUE to display the information.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, 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, does it generate a trigger event often enough?
       - Is the process name correct?
       - Is the process available and operational?
     - Can the queue be shared? If not, another application could already have it open for input.
     - Is the queue enabled appropriately for GET and PUT?

- If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the MQOPEN call has failed for some reason.

  Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. 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 IBM 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, that they 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 “Preparing channels” in the *MQSeries Intercommunication* book for information about starting channels.

Is your application or 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 mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

A performance problem 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 itself. If you suspect this, you need to contact your IBM Support Center for assistance.

Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making MQSeries calls are discussed in the following sections.

For more information about application design, see Chapter 2, “Overview of application design” in the *MQSeries Application Programming Guide*.

Effect of message length

The amount of data in a message can affect the performance of the application that processes the message. To achieve the best performance from your application, 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 100 MB, the maximum allowed by MQSeries.

Note also that if you do not set the *MaxMsgLength* attribute explicitly, it defaults to 4 MB, which may be very inefficient.

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.

Number of threads in use

For MQSeries for OS/2 Warp and MQSeries for Windows NT, an application may require a large number of threads. Each queue manager process is allocated a maximum allowable number of threads.

If some applications are troublesome, it could be due to their design using too many threads. Consider whether the application takes into account this possibility and that it takes actions either to stop or to report this type of occurrence.

The maximum number of threads that OS/2 allows is 4095. However, the default is 64. The default can be changed with the THREADS=xxxx parameter in CONFIG.SYS. MQSeries makes available up to 63 threads to its processes.

Incorrect output

The term “incorrect output” can be interpreted in many different ways. For the purpose of problem determination within this book, the meaning is explained in “Have you obtained incorrect output?” on page 252.

Two types of incorrect output are discussed in this section:

- Messages that do not appear when you are expecting them
- Messages that contain the wrong information, or information that has been corrupted

Additional problems that you might find if your application includes the use of distributed queues are also discussed.

Messages that do not appear on the queue

If messages do not appear when you are expecting them, check for the following:

- Has the message been put on the queue successfully?
  - Has the queue been defined correctly. For example, 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.
  - Has another application got exclusive access to the queue?
- Are you able to get any messages from the queue?
  - Do you need to take a syncpoint?

    If messages are being put or retrieved within syncpoint, they are not available to other tasks until the unit of recovery has been committed.
  - Is your wait interval long enough?

    You can set the wait interval as an option for the MQGET call. 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 whether you can get other messages from the queue.
  - Can other applications get messages from the queue?
  - Was the message you are expecting defined as persistent?

    If not, and 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 multiple server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If this is the case, refer to “Messages that contain unexpected or corrupted information” on page 258.

Messages that contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

- Has your application, or the application that put the message onto the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, the format of the message data may have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.

- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not really intended for an application servicing a different queue. If 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 connected to MQSeries. Check that the connection between the two systems is active.

  You can use an MQSeries PING command against either the queue manager (PING QMGR) or the channel (PING CHANNEL) to verify that the link is operable.

- Is triggering set on in the sending system?
- Is the message 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 “MQDLH - Dead-letter header” in 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 Intercommunication* book 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 format is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

  Refer to Chapter 11, “Writing data-conversion exits” in the *MQSeries Application Programming Guide* for further details of data conversion.

Error logs

MQSeries uses a number of error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use.

The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

In MQSeries for UNIX systems:

- If the queue manager name is known and the queue manager is available, error logs are located in:

  `/var/mqm/qmgrs/`*`qmname`*`/errors`

- If the queue manager is not available, error logs are located in:

  `/var/mqm/qmgrs/@SYSTEM/errors`

- If an error has occurred with a client application, error logs are located on the client’s root drive in

  `/var/mqm/errors`

In MQSeries for OS/2 Warp and Windows NT, and assuming that MQSeries has been installed on the C drive in the MQM directory:

- If the queue manager name is known and the queue manager is available, error logs are located in:

  `c:\mqm\qmgrs\`*`qmname`*`\errors`

- If the queue manager is not available, error logs are located in:

  `c:\mqm\qmgrs\@SYSTEM\errors`

- If an error has occurred with a client application, error logs are located on the client's root drive in:

  `c:\mqm\errors`

In MQSeries for Windows NT only, an indication of the error is also added to the Application Log, which can be examined with the Event Viewer application provided with Windows NT.

You can also examine the Registry to help resolve any errors. The Registry Editor supplied with Windows NT allows you to filter errors that are placed in the Event Log by placing the code in the following Registry entry:

`HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\IgnoredErrorCodes`

For example, to ignore error 5000, add AMQ5000 to the list.

Log files

At installation time an @SYSTEM errors subdirectory is created in the qmgrs file path. The errors subdirectory can contain up to three error log files named:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files have the same names as the @SYSTEM ones, that is AMQERR01, AMQERR02, and AMQERR03, and each has a capacity of 256 KB. The files are placed in the errors subdirectory of each queue manager that you create.

As error messages are generated, they are placed in AMQERR01. When AMQERR01 gets bigger than 256 KB it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager's errors files unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the @SYSTEM errors subdirectory.

To examine the contents of any error log file, use your usual system editor.

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 (/var/mqm or C:\MQM).

If the MQSeries configuration file is readable, and the DefaultPrefix attribute of the AllQueueManagers stanza is readable, errors are logged in the errors subdirectory of the directory identified by the DefaultPrefix attribute. For example, if the DefaultPrefix is C:\MQM, errors are logged in C:\MQM\ERRORS.

For further information about configuration files, see Chapter 11, “Configuring MQSeries” on page 127.

| **Note:** Errors in the Windows NT Registry are notified by messages when a queue
| manager is started.

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national language enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any. In addition, some operator messages are written to the AMQERR01.LOG file in the queue manager directory, and others to the @SYSTEM directory copy of the error log.

An example MQSeries error log

Figure 57 on page 262 shows a typical extract from an MQSeries error log.

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

Figure 57. Extract from an MQSeries error log

The MQSeries log-dump utility

For a description of the **dmpmqlog** command, see “dmpmqlog (Dump log)” on page 291.

Dead-letter queues

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the provided browse sample application (amqsbcg) to browse messages on the queue using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue. See “Browsing queues” on page 74 for more information about running this sample and about the kind of output it produces.

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

Problems may occur if you do not associate a dead-letter queue with each queue manager. For more information about dead-letter queues, see Chapter 12, “The MQSeries dead-letter queue handler” on page 157.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found, and result in “queue manager unavailable” type errors. Ensure that the configuration files exist, and that the MQSeries configuration file references the correct queue manager and log directories.

Note: Errors in the Windows NT Registry are notified by messages when a queue manager is started.

Tracing

This section describes how to produce a trace for each of the MQSeries Version 5 products.

Tracing MQSeries for AIX

MQSeries for AIX uses the standard AIX system trace. Tracing is a two-step process:

1. Gathering the data
2. Formatting the results

MQSeries uses two trace hook identifiers:

X'30D' This event is recorded by MQSeries on entry to or exit from a subroutine.

X'30E' This event is recorded by MQSeries to trace data such as that being sent or received across a communications network.

Trace provides detailed execution tracing to help you to analyze problems. IBM service support personnel may ask for a problem to be recreated with trace enabled. The files produced by trace can be **very** large so it is important to qualify a trace, where possible. For example, you can optionally qualify a trace by time and by component.

There are two ways to run trace:

1. **Interactively**

   The following sequence of commands runs an interactive trace on the program `myprog` and ends the trace.

```
trace -j30D,30E -o trace.file
->!myprog
->q
```

2. **Asynchronously**

   The following sequence of commands runs an asynchronous trace on the program `myprog` and ends the trace.

```
trace -a -j30D,30E -o trace.file
myprog
trcstop
```

You can format the trace file with the command:

```
trcrpt -t mqmtop/lib/amqtrc.fmt trace.file > report.file
```

`report.file` is the name of the file where you want to put the formatted trace output.

Note: **All** MQSeries activity on the machine is traced while the trace is active.

Selective component tracing

You should set the environment variable MQS_TRACE_OPTIONS only if you have been instructed to do so by your service personnel.

The environment variable MQS_TRACE_OPTIONS can be used to activate the high detail and parameter tracing functions individually. Because it enables tracing to be active without these functions, you can use it to reduce the overhead on execution speed when you are trying to reproduce a problem with tracing switched on. Table 17 on page 265 defines the trace behavior under the various settings of MQS_TRACE_OPTIONS.

Table 17. MQS_TRACE_OPTIONS settings	
MQS_TRACE_OPTIONS Value	**What will be traced**
Unset (default)	Default trace (all except high detail)
0	No MQSeries trace
262148	Entry, exit and parameter trace
786436	Entry, exit, parameter, and high detail trace
3407871	Default trace without parameter trace
3670015	Default trace, including parameter trace
4194303	All tracing, including high detail trace

Notes:

1. Typically MQS_TRACE_OPTIONS must be set in the process that starts the queue manager, and before the queue manager is started, or it is not recognized.
2. MQS_TRACE_OPTIONS must be set before tracing starts. If it is set after tracing starts it is not recognized.

An example of MQSeries for AIX trace data

The following example is an extract of an AIX trace:

```
 ...
ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

30D    1.189295104       0.000000   MQS FNC Exit.... 17726.1 xllListenSel
30D    1.189341184       0.046080   MQS CEI Entry... 17726.1 xllSpinLockR
30D    1.189364992       0.023808   MQS FNC Exit.... 17726.1 xllSpinLockR
30D    1.189380096       0.015104   MQS CEI Entry... 17726.1 xllSpinLockR
30D    1.189394816       0.014720   MQS FNC Exit.... 17726.1 xllSpinLockR
30D    1.189408512       0.013696   MQS CEI Entry... 17726.1 xllSpinLockR
30D    1.189427328       0.018816   MQS FNC Exit.... 17726.1 xllSpinLockR
30D    1.189444480       0.017152   MQS CEI Entry... 17726.1 xcsFreeQuick
30D    1.189461120       0.016640   MQS CEI Entry.... 17726.1 xllSpinLock
30D    1.189480320       0.019200   MQS FNC Exit..... 17726.1 xllSpinLock
30D    1.189592192       0.111872   MQS FNC Entry.... 17726.1 xstFreeCell
30D    1.189608448       0.016256   MQS FNC Exit..... 17726.1 xstFreeCell
30D    1.189658496       0.050048   MQS CEI Entry.... 17726.1 xllSpinLock
30D    1.189672832       0.014336   MQS FNC Exit..... 17726.1 xllSpinLock
30D    1.189691520       0.018688   MQS CEI Exit.... 17726.1 xcsFreeQuick
30D    1.189704064       0.012544   MQS CEI Entry... 17726.1 xllSpinLockR
30D    1.189717504       0.013440   MQS FNC Exit.... 17726.1 xllSpinLockR
30D    1.189729536       0.012032   MQS FNC Exit!.. 17726.1 xllWaitSocket
30D    1.189744512       0.014976   MQS FNC Exit!. 17726.1 xcsWaitEventSe
30D    1.189765376       0.020864   MQS CEI Exit! 17726.1 zcpReceiveOnLin
30D    1.189792128       0.026752   MQS FNC Entry 17726.1 zapInquireStatu
30D    1.189814400       0.022272   MQS FNC Entry. 17726.1 xcsRequestMute
30D    1.189832064       0.017664   MQS FNC Entry.. 17726.1 xllSemGetVal
30D    1.189898240       0.066176   MQS FNC Exit... 17726.1 xllSemGetVal
30E    1.204718976      14.820736   xcsRequestMutexSem phmtx:30000f3c Tim
 ...
```

Figure 58. Sample AIX trace

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.

Tracing MQSeries for HP-UX and MQSeries for Sun Solaris

In MQSeries for HP-UX and Sun Solaris, you enable or modify tracing using the **strmqtrc** control command, which is described in “strmqtrc (Start MQSeries trace)” on page 336. To stop tracing, you use the **endmqtrc** control command, which is described in “endmqtrc (End MQSeries trace)” on page 308. You can display formatted trace output using the **dspmqtrc** control command, which is described in “dspmqtrc (Display MQSeries formatted trace output)” on page 300.

Trace files

All trace files are created in the directory /var/mqm/trace.

Note: It is possible to accommodate production of large trace files by mounting a temporary file system over this directory.

Trace-file names have the following format:

AMQ*ppppp*.TRC

where *ppppp* is the process identifier (PID) of the process producing the trace.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

Example trace data

Figure 59 on page 267 shows an extract from an MQSeries for HP-UX trace.

```
 ...
 ID     ELAPSED_MICROSEC DELTA_MICROSEC  APPL    SYSCALL KERNEL  INTERRUP

30d     0                0          MQS FNC Exit........ 18855.1 xcsChec
30d     292              292        MQS CEI Entry....... 18855.1 xcsHSHM
30d     363              71         MQS CEI Exit........ 18855.1 xcsHSHM
30d     420              57         MQS CEI Entry....... 18855.1 xcsHSHM
30d     482              62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     539              57         MQS CEI Entry....... 18855.1 xcsHSHM
30d     602              63         MQS CEI Exit........ 18855.1 xcsHSHM
30d     659              57         MQS CEI Entry....... 18855.1 xcsHSHM
30d     721              62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     779              58         MQS CEI Entry....... 18855.1 xcsHSHM
30d     841              62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     899              58         MQS CEI Entry....... 18855.1 xcsHSHM
30d     961              62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     1018             57         MQS CEI Entry....... 18855.1 xcsHSHM
30d     1080             62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     1138             58         MQS CEI Entry....... 18855.1 xcsHSHM
30d     1200             62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     1257             57         MQS CEI Entry....... 18855.1 xcsHSHM
30d     1319             62         MQS CEI Exit........ 18855.1 xcsHSHM
30d     1377             58         MQS CEI Entry....... 18855.1 xcsHSHM
30d     1439             62         MQS CEI Exit........ 18855.1 xcsHSHME
30d     1498             59         MQS FNC Entry....... 18855.1 xcsAlloc
30d     1554             56         MQS CEI Entry........ 18855.1 xcsHSHM
30d     1616             62         MQS CEI Exit......... 18855.1 xcsHSHM
30d     1674             58         MQS FNC Entry........ 18855.1 xllSpin
30d     1733             59         MQS FNC Exit......... 18855.1 xllSpin
30e     1825             92         MQS  Signals Blocked with mask:
30e     1967             142        MQS Data from xcsAllocateQuickCell Le
            FFFFFBFF FFFFFFFF FFFFFFFF FFFFFFFF
            FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
 ...
```

Figure 59. Sample HP-UX trace

Figure 60 on page 268 shows an extract from an MQSeries for Sun Solaris trace.

```
 ...
ID     ELAPSED_MICROSEC DELTA_MICROSEC  APPL    SYSCALL KERNEL  INTERRUPT

30d     0                0          MQS FNC Exit.. 5814.1 xcsCheckProcess
30d     247              247        MQS CEI Entry. 5814.1 xcsHSHMEMBtoPTR
30d     301              54         MQS CEI Exit.. 5814.1 xcsHSHMEMBtoPTR
30d     343              42         MQS CEI Entry. 5814.1 xcsHSHMEMBtoPTR
30d     387              44         MQS CEI Exit.. 5814.1 xcsHSHMEMBtoPTR
30d     428              41         MQS CEI Entry. 5814.1 xcsHSHMEMBtoPTR
30d     472              44         MQS CEI Exit.. 5814.1 xcsHSHMEMBtoPTR
30d     514              42         MQS FNC Entry. 5814.1 xcsAllocateQuic
30d     554              40         MQS CEI Entry.. 5814.1 xcsHSHMEMBtoPT
30d     598              44         MQS CEI Exit... 5814.1 xcsHSHMEMBtoPT
30d     639              41         MQS FNC Entry.. 5814.1 xllSpinLockReq
30d     684              45         MQS FNC Exit... 5814.1 xllSpinLockReq
30e     764              80         MQS  Signals Blocked with mask:
30e     882              118        MQS Data from xcsAllocateQuickCell Le
            FFFFFFFF 00000FFF 00000000 00000000
30d     956              74         MQS CEI Entry.. 5814.1 xcsHSHMEMBtoPT
30d     1000             44         MQS CEI Exit... 5814.1 xcsHSHMEMBtoPT
30d     1040             40         MQS FNC Entry.. 5814.1 xstAllocateCel
30d     1082             42         MQS FNC Exit... 5814.1 xstAllocateCel
30e     1125             43         MQS  Signals Unblocked with mask:
30e     1222             97         MQS Data from xcsAllocateQuickCell Le
            00024007 00000000 00000000 00000000
30d     1373             151        MQS FNC Entry.. 5814.1 xllSpinLockRel
 ...
```

Figure 60. Sample MQSeries for Sun Solaris trace

Notes:

1. In these examples, 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.

Tracing MQSeries for OS/2 Warp and MQSeries for Windows NT

In MQSeries for OS/2 Warp and Windows NT, you enable or modify tracing using the **strmqtrc** control command, which is described in “strmqtrc (Start MQSeries trace)” on page 336. To stop tracing, you use the **endmqtrc** control command, which is described in “endmqtrc (End MQSeries trace)” on page 308.

For MQSeries for Windows NT Version 5.1 and later, you can also start and stop trace using the trace icon in the MQSeries Services snap-in.

Trace files

During the installation process, you can choose the drive on which trace files are to be located. However, the trace files are always placed in the directory \<mqmwork>\errors, where <mqmwork> is the directory selected when MQSeries was installed to hold MQSeries data files.

Trace-file names have the following format:

AMQ*ppppp*.TRC

where *ppppp* is the process identifier (PID) of the process producing the trace.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

An example of MQSeries for Windows NT trace data

Figure 61 shows an extract from an MQSeries for Windows NT trace:

```
        MQSeries Trace - Version 050000
        !! - BuildDate Sep 14 1998
 ...
00A2FE  ----> (0153) zcpCreatePacket
00A2FF  -----> (0153) xcsAllocateMemBlock
00A300  ------> (0153) xstAllocateMemBlock
00A301  -------> (0153) xstAllocBlockInSharedMemSet
00A302  --------> (0153) xstAllocBlockInAnExtentOnList
00A303  ---------> (0153) xstSerialiseExtentList
00A304  ----------> (0153) xllSpinLockRequest
00A305  -----------> (0153) xllAccessHandle
00A306  <----------- (0153) xllAccessHandle (rc=OK)
00A307  <---------- (0153) xllSpinLockRequest (rc=OK)
00A308  <--------- (0153) xstSerialiseExtentList (rc=OK)
00A309  ---------> (0153) xstAllocBlockInExtent
00A30A  ----------> (0153) xstSerialiseExtent
00A30B  -----------> (0153) xllSpinLockRequest
00A30C  ------------> (0153) xllAccessHandle
00A30D  <------------ (0153) xllAccessHandle (rc=OK)
00A30E  <----------- (0153) xllSpinLockRequest (rc=OK)
00A30F  <---------- (0153) xstSerialiseExtent (rc=OK)
00A310  ----------> (0153) xstInitialiseBlock
00A311  <---------- (0153) xstInitialiseBlock (rc=OK)
 ...
```

Figure 61. Sample MQSeries for Windows NT trace

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 (FFST)

This section describes the role of first-failure support technology (FFST) in each of the MQSeries Version 5.1 products.

FFST: MQSeries for UNIX systems

For MQSeries for UNIX systems, FFST information is recorded in a file in the /var/mqm/errors directory.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or an MQSeries internal error.

The files are named AMQ*nnnnn.mm*.FDC, where:

nnnnn	Is the ID of the process reporting the error
mm	Is a sequence number, normally 0

When a process creates an FFST record, it also sends a record to syslog. The record contains the name of the FFST file to assist in automatic problem tracking.

The syslog entry is made at the “user.error” level. See the operating-system documentation about syslog.conf for information about configuring this.

Some typical FFST data is shown in Figure 62.

```
MQSeries First Failure Symptom Report
=====================================

Date/Time         :- Friday August 14 19:51:59 BST 1998
Host Name         :- eclipse
PIDS              :- 5765B75
LVLS              :- 510
Product Long Name :- MQSeries for Sun Solaris 2 (Sparc)
Vendor            :- IBM
Probe Id          :- KN002003
Application Name  :- MQM
Component         :- kpiStartup
Build Date        :- Aug 13 1998 (Collector)
UserID            :- 00009126 (m8silk)
Program Name      :- amqzxma0
Process           :- 00000212
Thread            :- 00000001
QueueManager      :- wibble
Major Errorcode   :- arcE_CATALOGUE_NOT_FOUND
Minor Errorcode   :- OK
Probe Type        :- INCORROUT
Probe Severity    :- 2
Probe Description :- AMQ6125: An internal MQSeries error has occurred.


MQM Function Stack
kpiStartup
xcsFFST
```

Figure 62. FFST report for MQSeries for UNIX systems

The Function Stack and Trace History are used by IBM to assist in problem determination. In most cases there is little that the system administrator can do

when an FFST report is generated, apart from raising problems through the IBM Support Centers.

However, there are some problems that the system administrator might be able to solve. If the FFST shows “out of resource” or “out of space on device” descriptions when calling one of the IPC functions (for example, **semop** or **shmget**), it is likely that the relevant kernel parameter limit has been exceeded.

If the FFST report shows a problem with **setitimer**, it is likely that a change to the kernel timer parameters is needed.

To resolve these problems, increase the IPC limits, rebuild the kernel, and restart the machine. See one of the following for further information:

- “Chapter 2. Planning to Install the MQSeries for AIX Server” in the *MQSeries for AIX V5.1 Quick Beginnings* book
- “Chapter 2. Planning to Install the MQSeries for HP-UX Server” in the *MQSeries for HP-UX V5.1 Quick Beginnings* book
- “Chapter 2. Planning to Install the MQSeries for Sun Solaris Server” in the *MQSeries for Sun Solaris V5.1 Quick Beginnings* book

FFST: MQSeries for OS/2 Warp and Windows NT

In MQSeries for OS/2 Warp and Windows NT, FFST information is recorded in a file in the c:\mqm\errors directory.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or an MQSeries internal error.

FFST files are named AMQ*nnnnn*.*mm*.FDC, where:

nnnnn Is the ID of the process reporting the error
mm Is a sequence number, normally 0

When a process creates an FFST record it also sends a record to the Event Log. The record contains the name of the FFST file to assist in automatic problem tracking. The Event log entry is made at the “application” level.

A typical FFST log is shown in Figure 63 on page 272.

```
MQSeries First Failure Symptom Report
=====================================

Date/Time         :- Sun September 27 13:23:43 British Summer Time 1998
Host Name         :- HYRAX
PIDS              :- 5639B43
LVLS              :- 5100
Product Long Name :- MQSeries for Windows NT
Vendor            :- IBM
Probe Id          :- AD030001
Application Name  :- MQM
Component         :- adiReadFile
Build Date        :- Sep  9 1998
UserID            :- MQAdmin
Process Name      :- H:\MQM\BIN\amqzlaa0.exe
Process           :- 00000255
Thread            :- 00000273
QueueManager      :- x
Major Errorcode   :- xecF_E_UNEXPECTED_SYSTEM_RC
Minor Errorcode   :- OK
Probe Type        :- MSGAMQ6119
Probe Severity    :- 2
Probe Description :- AMQ6119: An internal MQSeries error has occurred
  (Rc=998 from ReadFile)
Comment1          :- Rc=998 from ReadFile



MQM Function Stack
zlaMainThread
zlaProcessMessage
...
adiReadFile
xcsFFST

MQM Trace History
                  <-- xihGetConnSPDetails rc=OK
                  <-- xllAccessMutexHandle rc=OK
```

Figure 63. Sample MQSeries for Windows NT First Failure Symptom Report

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

FFST: MQSeries for OS/2 Warp

In addition to the FFST files described in “FFST: MQSeries for OS/2 Warp and Windows NT” on page 271, MQSeries for OS/2 Warp also produces dumps in FFST/2™ format.

FFST/2 (First Failure Support Technology™ for OS/2) is an IBM licensed program that improves availability for IBM software applications by providing:

- Immediate event notification
- First failure data capture for software events
- Automated event tracking and management

FFST/2 is a corequisite to MQSeries for OS/2 Warp installation. At installation time, MQSeries checks whether FFST/2 exists on your system and, if so, at what level.

Refer to the *MQSeries for OS/2 Warp V5.1 Quick Beginnings* book for information about how FFST/2 is installed with MQSeries for OS/2 Warp.

Ensure that duplicate customized dumps are suppressed for FFST/2. This avoids the situation where you could receive multiple dumps when a software probe associated with FFST/2 is triggered more than once.

For information about using FFST/2, and how to suppress the duplicate dumps, refer to the *FFST/2 Administration Guide* (S96F-8593).

FFST/2 operation

There are a number of software probes included in the MQSeries code. These probes are triggered by specific events, usually error conditions. When a probe is triggered, the FFST/2 program can create the following output:

- A customized dump
- A symptom record
- A generic alert

You can use the diagnostic output to help you identify, track, and analyze an event. The diagnostic outputs are generated by default. You can override the defaults by making changes to the probe control table (PCT). For information about how to do this, refer to the *FFST/2 Administration Guide*.

MQSeries for OS/2 Warp entries in the PCT include the following fields:

Field	Description
program_id	5639B42
application	MQM
probe_id	A probe identifier that is set by MQSeries
options	The action that you want the FFST/2 program to take for the specified range of software probes

Using a symptom record

When an event triggers a software probe, the FFST/2 program creates a symptom record. When created, the symptom record can contain:

- The date and time the software probe was triggered
- Hardware and software vital product data
- Any error code information provided by the software probe
- The name of the dump file, if the software probe creates a customized dump file
- A unique problem identifier for each event
- The message number and the first 32 bytes of the message text, if the software probe specifies that a message is issued
- The primary symptom string, which uniquely identifies the event
- The secondary symptom string, if the software probe has one
- A brief description of the software probe, if one is available

The FFST/2 program places all symptom records in the OS/2 system error log. Parts of the symptom record are also included in the customized dump file. The maximum size of an entry in the OS/2 system error log is 4 KB. If the symptom record exceeds this limit, it is truncated.

To display the symptom records, display the system error log. This is accessed by opening the FFST/2 icon that is installed on your desktop and opening (double clicking on) the system error log. Note that after installation the FFST/2 icon could have been moved to another folder.

Using a customized dump

When the FFST/2 program creates a system dump for MQSeries following the triggering of a software probe, the dump file is named OS2SYSxx.DMP, where xx ranges from 00 to the number set when you configured FFST/2.

Whenever FFST/2 generates a dump, xx is incremented by 01. After xx reaches the maximum number of dumps, the FFST/2 program resets the number to 00 and begins overwriting existing dumps with new dumps.

If FFST/2 wraps, it adds a second batch of dump information to an already used dump file. When this dump file is viewed, the dump shown is of the oldest dump in the file. In order to view separate dumps within the dump file, you need to use the "identification" field. This will give a list of the problem identifiers stored in that dump file. You can then select the one you want to display.

For further information about FFST/2 records and how to use them, refer to the *FFST/2 Administration Guide*.

Problem determination with clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are additional reason codes for error conditions associated with clients. For example:

- Remote machine not responding
- Communications line error
- Invalid machine address

The most common time for errors to occur is when an application issues an MQCONN and receives the response MQRC_Q_MQR_NOT_AVAILABLE. An error message, written to the client log file, explains the cause of the error. Messages may also be logged at the server depending on the nature of the failure.

Terminating clients

Even though a client has terminated, it is still possible for the process at the server to be holding its queues open. Normally, this will only be for a short time until the communications layer detects that the partner has gone.

Error messages with clients

When an error occurs with a client system, error messages are put into the error files associated with the server, if possible. If an error cannot be placed there, the client code attempts to place the error message in an error log in the root directory of the client machine.

OS/2 and UNIX-systems clients

Error messages for OS/2 and UNIX 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 on UNIX systems.

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 the C drive. Working in the DOS environment involves the environment variable MQDATA.

This is the default library used by the client code to store trace and error information; it also holds the directory name in which the qm.ini file is stored. (needed for NetBIOS setup). If not specified, it defaults to the C drive.

Note: For Windows NT clients, the default library does not contain the directory name for the qm.ini file because configuration information is stored in the Windows NT Registry.

The names of the default files held in this library are:

AMQERR01.LOG For error messages.

AMQERR01.FDC For First Failure Data Capture messages.

For more information about clients, see the *MQSeries Clients* book.

Part 2. Reference

Chapter 17. MQSeries control commands

This chapter contains reference material for the control commands supported by MQSeries Version 5.1 products. Please note the following environment-specific information:

MQSeries for AIX
All commands in this chapter can be issued from an AIX shell. These commands are case-sensitive.

MQSeries for HP-UX
All commands in this chapter can be issued from an HP-UX shell. These commands are case sensitive.

MQSeries for OS/2 Warp
All commands in this chapter can be issued from a command line. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case sensitive.

In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.

MQSeries for Sun Solaris
All commands in this chapter can be issued from a Solaris shell. These commands are case sensitive.

MQSeries for Windows NT
All commands in this chapter can be issued from a command line. A subset can be issued using the MQSeries Explorer snap-in. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case sensitive.

In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.

Names of MQSeries objects

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
- Namelists
- Clusters

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 1)
- Percent sign (%) (see note 1)

Notes:

1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.
2. Leading or embedded blanks are not allowed.
3. National language characters are not allowed.
4. Names may be enclosed in double quotation marks, but this is essential only if special characters are included in the name.

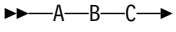
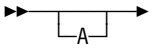
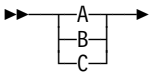
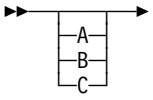
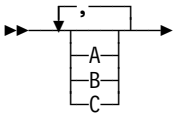
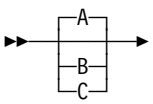
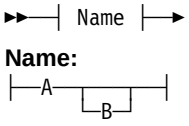
How to read syntax diagrams

This 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.

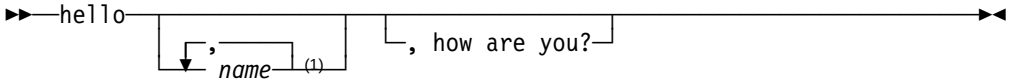
Other conventions used in syntax diagrams are:

Table 18. How to read syntax diagrams

Convention	Meaning
	You must specify values `A`, `B`, and `C`. Required values are shown on the main line of a syntax diagram.
	You may specify value `A`. Optional values are shown below the main line of a syntax diagram.
	Values `A`, `B`, and `C` are alternatives, one of which you must specify.
	Values `A`, `B`, and `C` are alternatives, one of which you may specify.
	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:



Note:
1 You can code up to three names.

According to the syntax diagram, these are all valid versions of the **hello** command:

```
hello
hello name
hello name, name
hello name, name, name
hello, how are you?
hello name, how are you?
hello name, name, how are you?
hello name, name, name, how are you?
```

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 for 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. Different forms of parentheses are used to indicate whether a parameter is required. For example:

```
CmdName [-x OptParam ] ( -c | -b ) argument
```

where:

`CmdName`	Is the command name for which help has been requested.
`[-x OptParam ]`	Square brackets enclose one or more optional parameters. Where square brackets enclose multiple parameters, you can select no more than one of them.
`( -c | -b )`	Brackets enclose multiple values, one of which you must select. In this example, you must select either flag c or flag b.
`argument`	A mandatory argument.

Examples

1. Result of entering `endmqm ?`

   ```
   endmqm [-z][-c | -w | -i | -p] QMgrName
   ```

2. Result of entering `rcdmqimg ?`

   ```
   rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]
   ```

crtmqcvx (Data conversion)

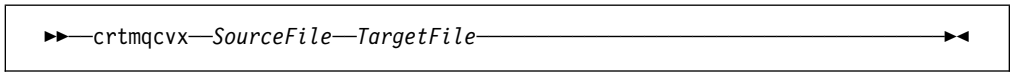
Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert C structures.

The command reads an input file containing structures to be converted, and writes an output file containing code fragments to convert those structures.

For information about using this command, see "Writing a data-conversion exit program" in the *MQSeries Application Programming Guide*.

Syntax



Required 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, be aware that the fragment uses macros supplied in the MQSeries header file `amqsvmha.h`.

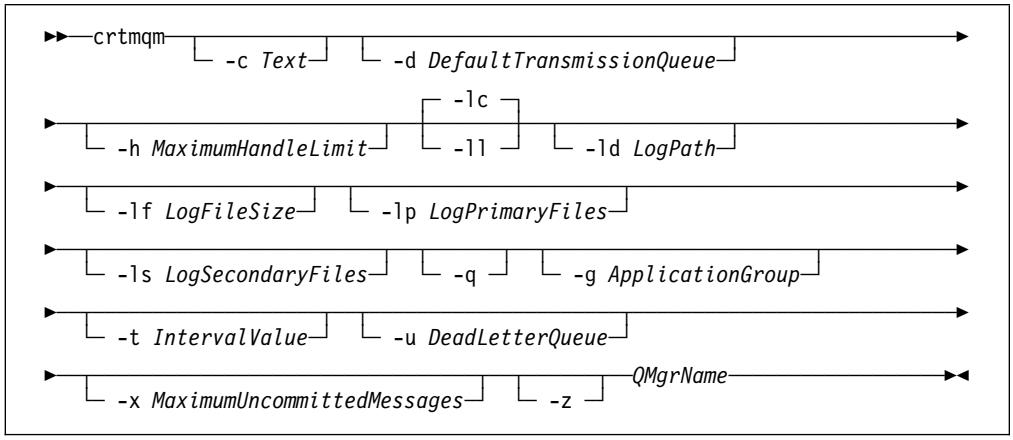
```
MQLONG Convertmy_structure(
              PMQBYTE *in_cursor,
              PMQBYTE *out_cursor,
              PMQBYTE in_lastbyte,
              PMQBYTE out_lastbyte,
              MQHCONN hConn,
              MQLONG  opts,
              MQLONG  MsgEncoding,
              MQLONG  ReqEncoding,
              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 and define the default and system objects. The objects created by **crtmqm** are listed in Appendix A, “System and default objects” on page 343. When a queue manager has been created, use the **strmqm** command to start it.

Syntax



Required 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*

Specifies some descriptive text for this queue manager. You can use up to 64 characters; the default is all blanks.

If special characters are required, the description must be enclosed in double quotes. The maximum number of characters is reduced if the system is using a double-byte character set (DBCS).

-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 1 through 999 999 999. The default value is 256.

The next six parameter descriptions relate to logging, which is described in “Using the log for recovery” on page 222.

-lc Circular logging is to be used. This is the default logging method.

-ll Linear logging is to be used.

-ld *LogPath*
Specifies the directory to be used to hold log files.

In MQSeries for UNIX systems, the default is /var/mqm/log.

User ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This occurs automatically if the log files are in their default locations.

In MQSeries for OS/2 Warp and Windows NT, the default is C:\MQM\LOG (assuming that C is your data drive).

-lf *LogFileSize*
Specifies the size of the log files in units of 4 KB.

In MQSeries for UNIX systems, the minimum value is 64, and the maximum is 16384. The default value is 1024, giving a default log size of 4 MB.

In MQSeries for OS/2 Warp and Windows NT, the minimum value is 32, and the maximum is 4095. The default value is 256, giving a default log size of 1 MB.

-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.

The limits given in the previous parameter descriptions are limits set by MQSeries. Operating system limits may reduce the maximum possible log size.

-q Specifies that this queue manager is to be made the default queue manager. The new queue manager replaces any existing default queue manager.

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 11, “Configuring MQSeries” on page 127 for information about configuration files.

-g *ApplicationGroup*

Specifies the name of the group whose members are allowed to:

- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories

This option applies only to MQSeries for AIX, Sun Solaris, and HP-UX.

The default value is -g all, which allows unrestricted access.

The -g *ApplicationGroup* value is recorded in the queue manager configuration file, qm.ini.

| The mqm user ID **must** belong to the specified ApplicationGroup.

-t *IntervalValue*

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 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, triggering can be enabled again 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 1 through 999 999 999. The default value is
| 10 000 uncommitted messages.

-z Suppresses error messages.

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.

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`, and creates the system and default objects. It also specifies that linear logging is to be used:

```
crtmqm -c "Paint shop" -ll -q Paint.queue.manager
```

2. This command creates a default queue manager named `Paint.queue.manager`, creates the system and default objects, and requests two primary and three secondary log files:

```
crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
```

3. This command creates a queue manager called `travel`, creates the system and default objects, sets the trigger interval to 5000 milliseconds (or 5 seconds), and specifies SYSTEM.DEAD.LETTER.QUEUE as its dead-letter queue.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
```

Related commands

strmqm	Start queue manager
endmqm	End queue manager
dltmqm	Delete queue manager

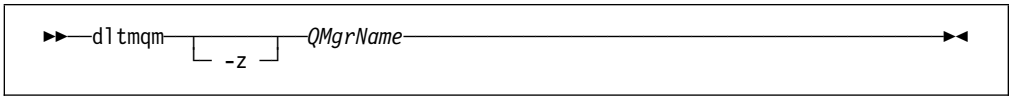
dltmqm (Delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager. 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.

In MQSeries for Windows NT, if you attempt to delete a queue manager when queue manager files are open, an error may occur. In this case, close the files and reissue the command.

Syntax



Required parameters

QMgrName
Specifies the name of the queue manager to be deleted.

Optional parameters

-z Suppresses error messages.

Return codes

0	Queue manager deleted
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
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

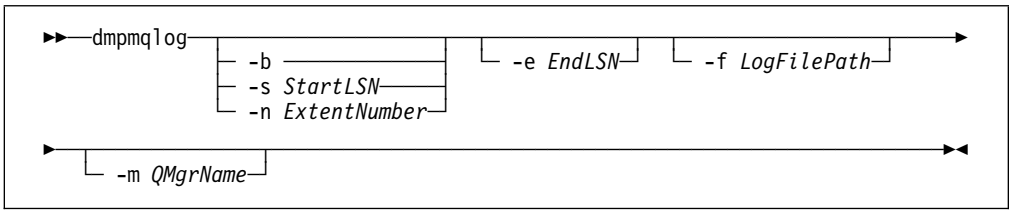
dmpmqlog (Dump log)

Purpose

Use the **dmpmqlog** command to dump a formatted version of the MQSeries system log.

The log to be dumped must have been created on the same type of operating system as that being used to issue the command.

Syntax



Optional parameters

Dump start point

Use one of the following parameters to specify the log sequence number (LSN) at which the dump should start. If no start point is specified, dumping starts by default from the LSN of the first record in the active portion of the log.

`-b` Specifies that dumping should start from the base LSN. The base LSN identifies the start of the log extent that contains the start of the active portion of the log.

`-s` *StartLSN*
Specifies that dumping should start from the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

If you are using a circular log, the LSN value must be equal to or greater than the base LSN value of the log.

`-n` *ExtentNumber*
Specifies that dumping should start from the specified extent number. The extent number must be in the range 0–9 999 999.

This parameter is valid only for queue managers whose *LogType* (as recorded in the configuration file, qm.ini) is LINEAR.

`-e` *EndLSN*
Specifies that dumping should end at the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

`-f` *LogFilePath*

Is the absolute, rather than the relative, directory path name to the log files. The specified directory must contain the log header file (`amqhlctl.lfh`) and a subdirectory called `active`. The active subdirectory must contain the log files. By default, log files are assumed to be in the directories specified in the mqs.ini and qm.ini files. If this option is used then queue names, associated with queue identifiers, will only be shown in the dump if a queue manager name is specified explicitly for the -m option and that queue manager has the object catalog file in its directory path.

On a system that supports long file names this file is named qmqmobjcat and, in order to map the queue identifiers to queue names, it must be the file used when the log files were created. As an example, for a queue manager named qm1, the object catalog file is located in the directory ..\qmgrs\qm1\qmanager\. To achieve this mapping, it may be necessary to create a temporary queue manager, for example named tmpq, replace its object catalog with the one associated with the specific log files, and then start dmpmqlog, specifying `-m tmpq` and `-f` with the absolute directory path name to the log files.

`-m` *QMgrName*

Is the name of the queue manager. If this parameter is omitted, the name of the default queue manager is used.

The queue manager you specify, or default to, must not be running when the **dmpmqlog** command is issued. Similarly, the queue manager must not be started while **dmpmqlog** is running.

dspmqaut (Display authority)

Purpose

Use the **dspmqaut** command to display the current authorizations to a specified object.

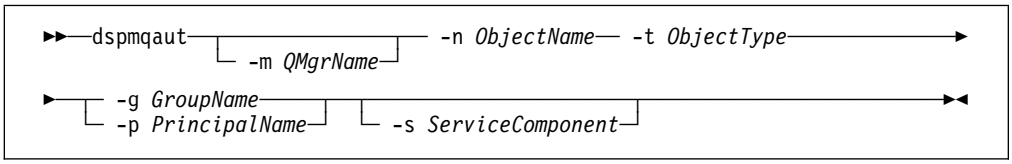
If a user ID is a member of more than one group, this command displays the combined authorizations of all of the groups.

Only one group or principal may be specified.

You can use this command meaningfully in MQSeries for OS/2 Warp only if an authorization service component has been installed for the current queue manager. For MQSeries for OS/2 Warp, this does not occur by default, and no such component is supplied with the product. Furthermore, when you issue this command, the results always indicate that any group or principal has all authorizations.

For more information about authorization service components, see "The Service stanza" on page 136 and "The ServiceComponent stanza" on page 137 in this book, and the *MQSeries Programmable System Management* book.

Syntax



Required parameters

-n *ObjectName*

Specifies the name of a queue manager, queue, or process definition on which the inquiry is to be made.

This is a required parameter, ***unless*** the inquiry relates to the queue manager itself, in which case it must not be included.

-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 type parameter
qmgr	A queue manager object
process or **prcs**	A process
namelist or **nl**	A namelist

Optional parameters

-m *QMgrName*
Specifies the name of the queue manager on which the inquiry is to be made.

-g *GroupName*
Specifies the name of the user group on which the inquiry is to be made. You can specify only 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.

For MQSeries for Windows NT only, the name of the principal can optionally include a domain name which should be specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see “Using the @ symbol in user ID names” on page 106.

-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 inquiry 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 values. Each authorization value returned means that any user ID in the specified group has the authority to perform the operation defined by that value.

Table 19 on page 295 shows the authorities that can be given to the different object types.

Table 19. Security authorities from the dspmqaut command

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

The following list defines the authorizations associated with each value:

all
: Use all operations relevant to the object.

alladm
: Perform all administration operations relevant to the object.

allmqi
: Use all MQI calls relevant to the object.

altusr
: Specify an alternate user ID on an MQI call.

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

chg
: Change the attributes of the specified object, using the appropriate command set.

clr
: Clear a queue (PCF command Clear queue only).

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

crt
: Create objects of the specified type, using the appropriate command set.

dlt
: Delete the specified object, using the appropriate command set.

dsp
: Display the attributes of the specified object, using the appropriate command set.

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

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

passall
: Pass all context.

passid
: Pass the identity context.

put
: Put a message on a specific queue by issuing an MQPUT call.

set
: Set attributes on a queue from the MQI by issuing an MQSET call.

setall
: Set all context on a queue.

setid
: Set the identity context on a queue.

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands
- MQSC commands
- PCF commands

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing

Examples

The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

```
dspmqaut -m saturn.queue.manager -t qmgr -g staff
```

The results from this command are:

```
Entity staff has the following authorizations for object :
        get
        browse
        put
        inq
        set
        connect
        altusr
        passid
        passall
        setid
```

Related commands

setmqaut	Set or reset authority

dspmqcsv (Display command server)

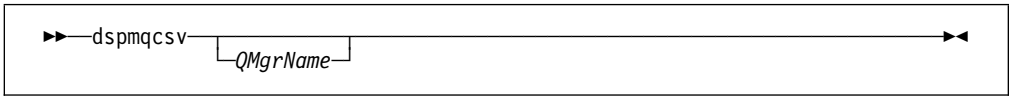
Purpose

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

Syntax



Required parameters

None

Optional 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

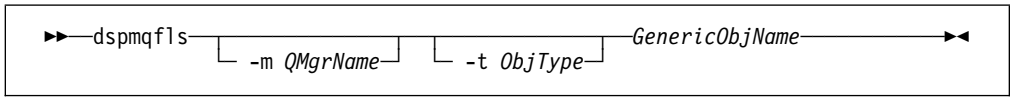
strmqcsv	Start a command server
endmqcsv	End a command server

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 “Understanding MQSeries file names” on page 27 for further information about name transformation.

Syntax



Required 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
ctlg or **catalog**	An object catalog
nl or **namelist**	A namelist

Notes:

1. The **dspmqfls** command displays the directory containing the queue, ***not*** the name of the queue itself.
2. In MQSeries for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, '*'. To accomplish this, use 'quoting'.

   There are a number of ways of 'quoting', depending on your shell. For example, single quotation 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)

Special note

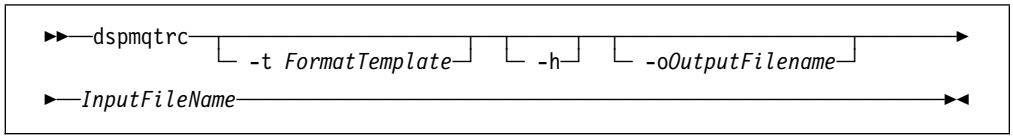
The **dspmqtrc** command is not supported by these MQSeries products:

- MQSeries for AIX
- MQSeries for OS/2 Warp
- MQSeries for Windows NT

Purpose

Use the **dspmqtrc** command to display MQSeries formatted trace output.

Syntax



Required parameters

InputFileName
Specifies the name of the file containing the unformatted trace. For example `/var/mqm/trace/AMQ12345.TRC`.

Optional parameters

-t *FormatTemplate*
Specifies the name of the template file containing details of how to display the trace. The default value is **mqmtop**`/lib/amqtrc.fmt`.

-h Omit header information from the report.

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

Related commands

endmqtrc	End MQSeries trace
strmqtrc	Start MQSeries trace

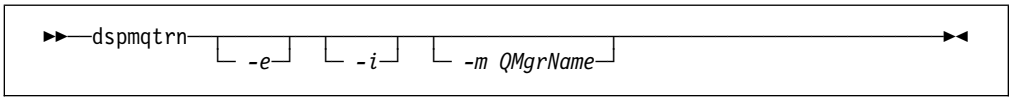
dspmqtrn (Display MQSeries transactions)

Purpose

Use the **dspmqtrn** command to display details of in-doubt transactions. Such transactions can be either internally or externally coordinated.

For each in-doubt transaction, a transaction number (a human-readable transaction identifier), the transaction state, and the transaction ID are displayed. (Transaction IDs can be up to 128 characters long, hence the need for a transaction number.)

Syntax



Optional parameters

-e Requests details of externally coordinated, in-doubt transactions. Such transactions are those for which MQSeries has been asked to prepare to commit, but has not yet been informed of the transaction outcome.

-i Requests details of internally coordinated, in-doubt transactions. Such transactions are those for which each resource manager has been asked to prepare to commit, but MQSeries has yet to inform the resource managers of the transaction outcome.

Information about the (deduced) state of the transaction in each of its participating resource managers is displayed. This information can help you assess the effects of failure in a particular resource manager.

-m *QMgrName*
Specifies the name of the queue manager whose transactions are to be displayed. If no name is specified, the default queue manager's transactions are displayed.

Note: If you specify neither -e nor -i, details of both internally and externally coordinated in-doubt transactions are displayed.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

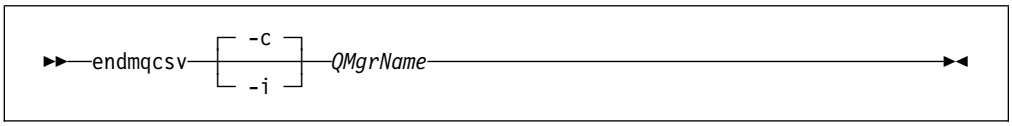
rsvmqtrn	Resolve MQSeries transaction

endmqcsv (End command server)

Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.

Syntax



Required 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

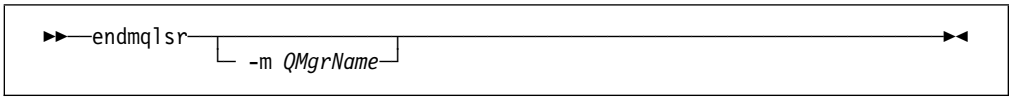
endmqlsr (End listener)

Purpose

The **endmqlsr** command ends all listener process for the specified queue manager.

The queue manager must be stopped before the **endmqlsr** command is issued.

Syntax



Optional parameters

-m *QMgrName*
Specifies the name of the queue manager. If no name is specified, the processing will be done for the default queue manager.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

endmqm (End queue manager)

Purpose

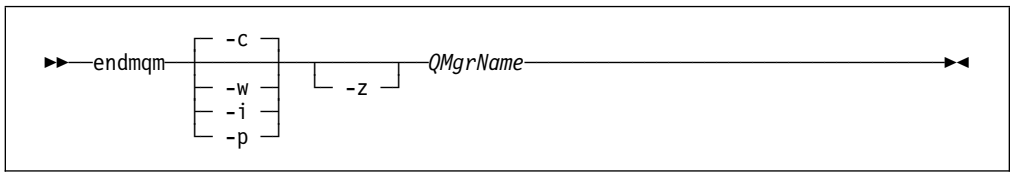
Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Controlled or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, you must stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Required parameters

QMgrName
Specifies the name of the message queue manager to be stopped.

Optional parameters

-c Controlled (or quiesced) shutdown. This is the default.

The queue manager stops but only after all applications have disconnected. Any MQI calls currently being processed are completed.

Control is returned to you immediately and you are not notified of when the queue manager has stopped.

-w Wait shutdown.

This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped. You receive the message “Waiting for queue manager `qmName` to end” while shutdown progresses.

-i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

Control is returned after the queue manager has ended.

-p Preemptive shutdown.

Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for MQSeries applications. All processes in the queue manager that fail to stop are terminated 30 seconds after the command is issued.

-z Suppresses error messages on the command.

Return codes

0	Queue manager ended
3	Queue manager being created
16	Queue manager does not exist
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error

Examples

The following examples show commands that end (stop) the specified queue managers.

1. This command ends the queue manager named `mercury.queue.manager` in a controlled way. All applications currently connected are allowed to disconnect.

```
endmqm mercury.queue.manager
```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

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

Related commands

crtmqm	Create a queue manager
strmqm	Start a queue manager
dltmqm	Delete a queue manager

endmqtrc (End MQSeries trace)

Special note

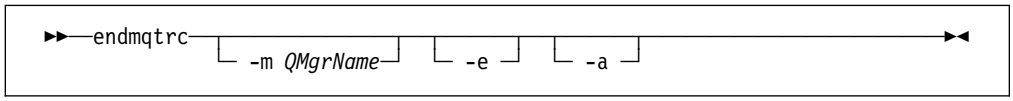
The **endmqtrc** command is not supported by MQSeries for AIX.

Purpose

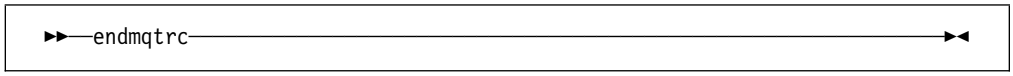
Use the **endmqtrc** command to end tracing for the specified entity or all entities.

Syntax

The syntax of this command in MQSeries for HP-UX and Sun Solaris is as follows:



The syntax of this command in MQSeries for OS/2 Warp and Windows NT is as follows:



Optional parameters

The following parameters can be specified in MQSeries for HP-UX and Sun Solaris only:

-m *QMgrName*
Is the name of the queue manager for which tracing is to be ended.

A maximum of one -m flag and associated queue manager name can be supplied on the command.

A queue manager name and -m flag can be specified on the same command as the -e flag.

-e 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

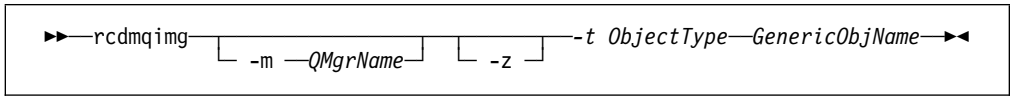
rcdmqimg (Record media image)

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.

Syntax



Required parameters

GenericObjName

Specifies the name of the object that is to be recorded. This parameter can 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 is required ***unless*** you are recording a queue manager object or the channel synchronization file. If you specify an object name for the channel synchronization file, it is ignored.

-t *ObjectType*

Specifies the types of object whose images are to be recorded. Valid object types are:

nl or **namelist**	Namelists
prcs or **process**	Processes
q or **queue**	All types of queue
ql or **qlocal**	Local queues
qa or **qalias**	Alias queues
qr or **qremote**	Remote queues
qm or **qmodel**	Model queues
qmgr	Queue manager object
syncfile	Channel synchronization file
ctlg or **catalog**	An object catalog
***** or **all**	All of the above

Note: When using MQSeries for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, '*'. To accomplish this, use 'quoting'. There are a number of ways of 'quoting' depending on your shell. For example, single quotation 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 Suppresses error messages.

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.

```
rcdmqimg -t qmgr -m saturn.queue.manager
```

Related commands

rcrmqobj	Recreate a queue manager object

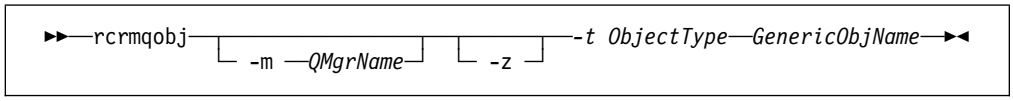
rcrmqobj (Recreate object)

Purpose

Use the **rcrmqobj** command to recreate an object, or group of objects, from their images contained in the log. Use the associated command, **rcdmqimg**, to record the object images to the log.

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.

Syntax



Required parameters

GenericObjName

Specifies the name of the object that is to be recreated. This parameter can 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.

This parameter is required ***unless*** the object type is the channel synchronization file; if an object name is supplied for this object type, it is ignored.

-t ObjectType

Specifies the types of object to be recreated. Valid object types are:

nl or **namelist**	Namelist
prcs or **process**	Processes
q or **queue**	All types of queue
ql or **qlocal**	Local queues
qa or **qalias**	Alias queues
qr or **qremote**	Remote queues
qm or **qmodel**	Model queues
syncfile	The channel synchronization file
***** or **all**	All the above

Note: When using MQSeries for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, '*'. To accomplish this, use 'quoting'. There are a number of ways of 'quoting' depending on your shell. For example, single quotation 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 Suppresses error messages.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
66	Media image not available
68	Media recovery 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 Record an MQSeries object in the log

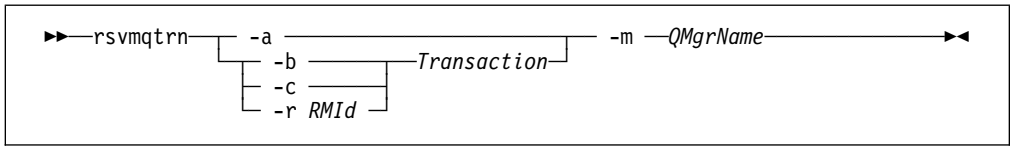
rsvmqtrn (Resolve MQSeries transactions)

Purpose

Use the **rsvmqtrn** command to commit or back out internally or externally coordinated in-doubt transactions.

Use this command only when you are certain that transactions cannot be resolved by the normal protocols. Issuing this command may result in the loss of transactional integrity between resource managers for a distributed transaction.

Syntax



Required parameters

-m *QMgrName*
Specifies the name of the queue manager. This parameter is mandatory.

Optional parameters

-a Specifies that the queue manager should attempt to resolve all internally coordinated, in-doubt transactions (that is, all global units of work).

-b Specifies that the named transaction is to be backed out. This flag is valid for externally coordinated transactions (that is, for external units of work) only.

-c Specifies that the named transaction is to be committed. This flag is valid for externally coordinated transactions (that is, for external units of work) only.

-r *RMId*
Identifies the resource manager to which the commit or back out decision applies. This flag is valid for internally coordinated transactions only, and for resource managers that are no longer configured in the queue manager's qm.ini file[5] The outcome delivered will be consistent with the decision reached by MQSeries for the transaction.

Transaction
Is the transaction number of the transaction being committed or backed out. To discover the relevant transaction number, use the **dspmqtrn** command. This parameter is required with the -b, -c, and -r *RMId* parameters.

[5] The Windows NT Registry for MQSeries for Windows NT Version 5.1 and later.

Return codes

0	Successful operation
32	Transactions could not be resolved
34	Resource manager not recognized
35	Resource manager not permanently unavailable
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

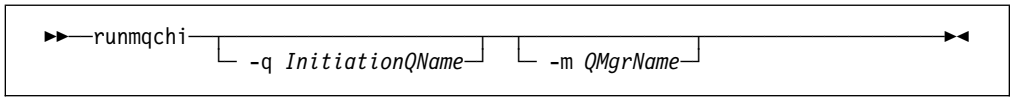
dspmqtrn	Display list of prepared transactions

runmqchi (Run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to Chapter 9, “Preparing MQSeries for distributed platforms” in the *MQSeries Intercommunication* book.

Syntax



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” on page 259.

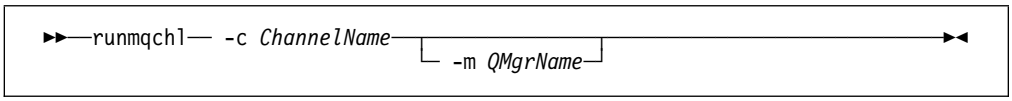
runmqchl (Run channel)

Purpose

Use the **runmqchl** command to run either a Sender (SDR) or a Requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax



Required parameters

-c *ChannelName*
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

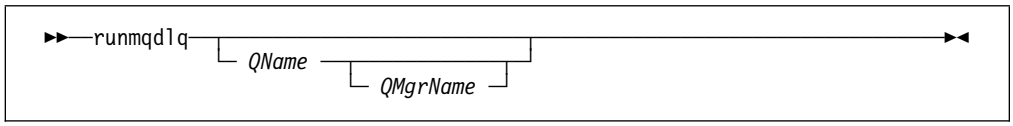
If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages. You should also review the @SYSTEM error log because problems that occur before the channel is associated with the queue manager are recorded there.

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.

Syntax



Description

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. In MQSeries for AIX, HP-UX, and Sun Solaris, it does not start to process the named queue until it receives an end_of_file (Ctrl+D) character. In MQSeries for OS/2 Warp and Windows NT, it does not start to process the named queue until exactly the following key sequence is pressed: Ctrl+Z, Enter, Ctrl+Z, Enter.

For more information about rules tables and how to construct them, see “The DLQ handler rules table” on page 158.

Note: For MQSeries for OS/2 Warp, the final rule must be terminated by an end-of-line character.

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, you could 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.

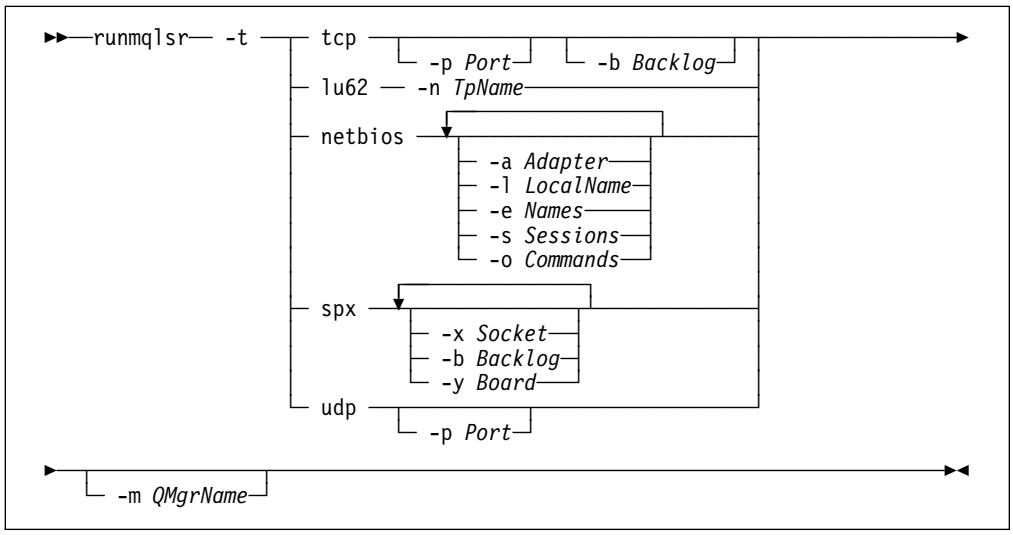
If no name is specified, the default queue manager for the installation is used. If one or more blanks (' ') are used, the default queue manager for this installation is explicitly assigned.

runmqlsr (Run listener)

Purpose

The **runmqlsr** (Run listener) command runs a listener process.

Syntax



Required parameters

`-t` Specifies the transmission protocol to be used:

`tcp` Transmission Control Protocol / Internet Protocol (TCP/IP)

`lu62` SNA LU 6.2

`netbios` NetBIOS

`spx` SPX

`udp` User datagram protocol (UDP) for AIX platforms only

Optional parameters

`-p` *Port* Port number for TCP/IP. This flag is valid for TCP and UDP. If a value is not specified, the value is taken from the queue manager configuration file, or from defaults in the program. The default value is 1414.

`-n` *TpName*

LU 6.2 transaction program name. This flag is valid only for the LU 6.2 transmission protocol. If a value is not specified, the value is taken from the queue manager configuration file. If a value is not given, the command fails.

`-a` *Adapter*

Specifies the adapter number on which NetBIOS listens. The default value is 0, that is, the listener uses adapter 0.

-l *LocalName*
: Specifies the NetBIOS local name that the listener uses. The default is specified in the queue manager configuration file.

-e *Names*
: Specifies the number of names that the listener can use. The default value is specified in the queue manager configuration file, qm.ini.

-s *Sessions*
: Specifies the number of sessions that the listener can use. The default value is specified in the queue manager configuration file, qm.ini.

-o *Commands*
: Specifies the number of commands that the listener can use. The default value is specified in the queue manager configuration file, qm.ini.

-x *Socket*
: Specifies the SPX socket on which SPX listens. The default value is hexadecimal 5E86.

-y *Board*
: Specifies the adapter number for SPX.

  This parameter applies to MQSeries for OS/2 Warp only.

-m *QMgrName*
: Specifies the name of the queue manager. If no name is specified, the command operates on the default queue manager.

-b *Backlog*
: Specifies the number of concurrent connection requests that the listener supports. See “The LU62, NETBIOS, TCP, and SPX stanzas” on page 144 for a list of default values and further information.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. Five names, five commands, and five sessions are specified for this listener, indicating the maximum number of each that this listener can use. These resources must be within the limits set in the queue manager configuration file, qm.ini.

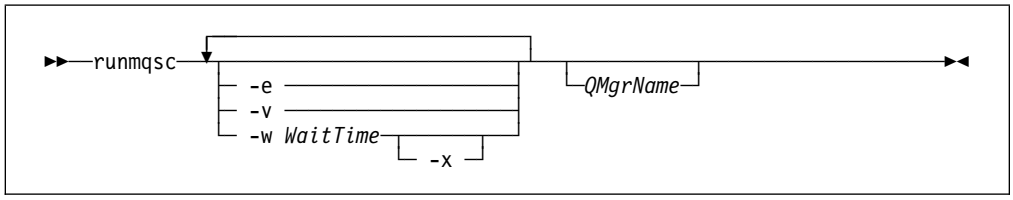
```
runmqlsr -t netbios -e 5 -s 5 -o 5
```

runmqsc (Run MQSeries commands)

Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks, for example defining, altering, or deleting a local queue object. MQSC commands and their syntax are described in Chapter 2, “The MQSeries commands” in the *MQSeries Command Reference* manual.

Syntax



Description

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 available only 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.

Indirect mode operation is performed through the default queue manager.

The **runmqsc** command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

Note: To run this command in MQSeries for UNIX systems, your 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; this verifies the specified commands without performing the actions. This mode is only available locally. The -w and -x flags are ignored if they are specified at the same time.

-w *WaitTime*
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” on page 90 for more information.

WaitTime Specifies the time, in seconds, that **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 OS/390. 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 OS/390 command queue.

QMgrName
Specifies the name of the target queue manager on which the MQSC commands 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. Enter this command at the command prompt:

```
runmqsc
```

   Now you can enter MQSC commands directly at the command prompt. No queue manager name is specified, therefore the MQSC commands are processed on the default queue manager.

2. Use one of these commands, as appropriate in your environment, to specify that MQSC commands are to be verified only:

```
runmqsc -v BANK < /u/users/commfile.in

runmqsc -v BANK < c:\users\commfile.in
```

   This command verifies the MQSC commands in file `commfile.in`. The queue manager name is `BANK`. The output is displayed in the current window.

3. These commands run the MQSC command file `mqscfile.in` against the default queue manager.

```
runmqsc < /var/mqm/mqsc/mqscfile.in > /var/mqm/mqsc/mqscfile.out

runmqsc < c:\mqm\mqsc\mqscfile.in > c:\mqm\mqsc\mqscfile.out
```

   In this example, the output is directed to file `mqscfile.out`.

runmqtmc (Start client trigger monitor)

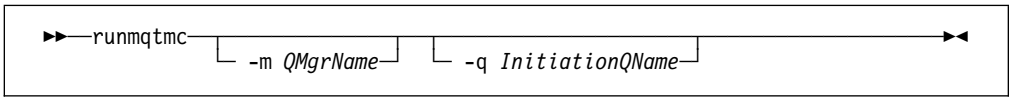
Special note

The **runmqtmc** command is available on OS/2 and AIX clients only.

Purpose

Use the **runmqtmc** command to invoke a trigger monitor for a client. For further information about using trigger monitors, refer to “Trigger monitors” in the *MQSeries Application Programming Guide*.

Syntax



Optional parameters

-m *QMgrName*
Specifies the name of the queue manager on which the client trigger monitor operates. If the name is omitted, the client trigger monitor operates on the default queue manager.

-q *InitiationQName*
Specifies the name of the initiation queue to be processed. If the name is 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.

Examples

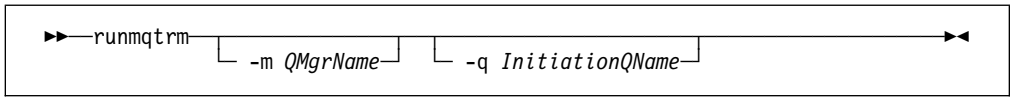
For examples of the use of this command, refer to the *MQSeries Application Programming Guide*.

runmqtrm (Start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to “Trigger monitors” in the *MQSeries Application Programming Guide*.

Syntax



Optional parameters

-m *QMgrName*
Specifies the name of the queue manager on which the trigger monitor operates. If the name is omitted, the trigger monitor operates on the default queue manager.

-q *InitiationQName*
Specifies the name of the initiation queue to be processed. If the name is 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)

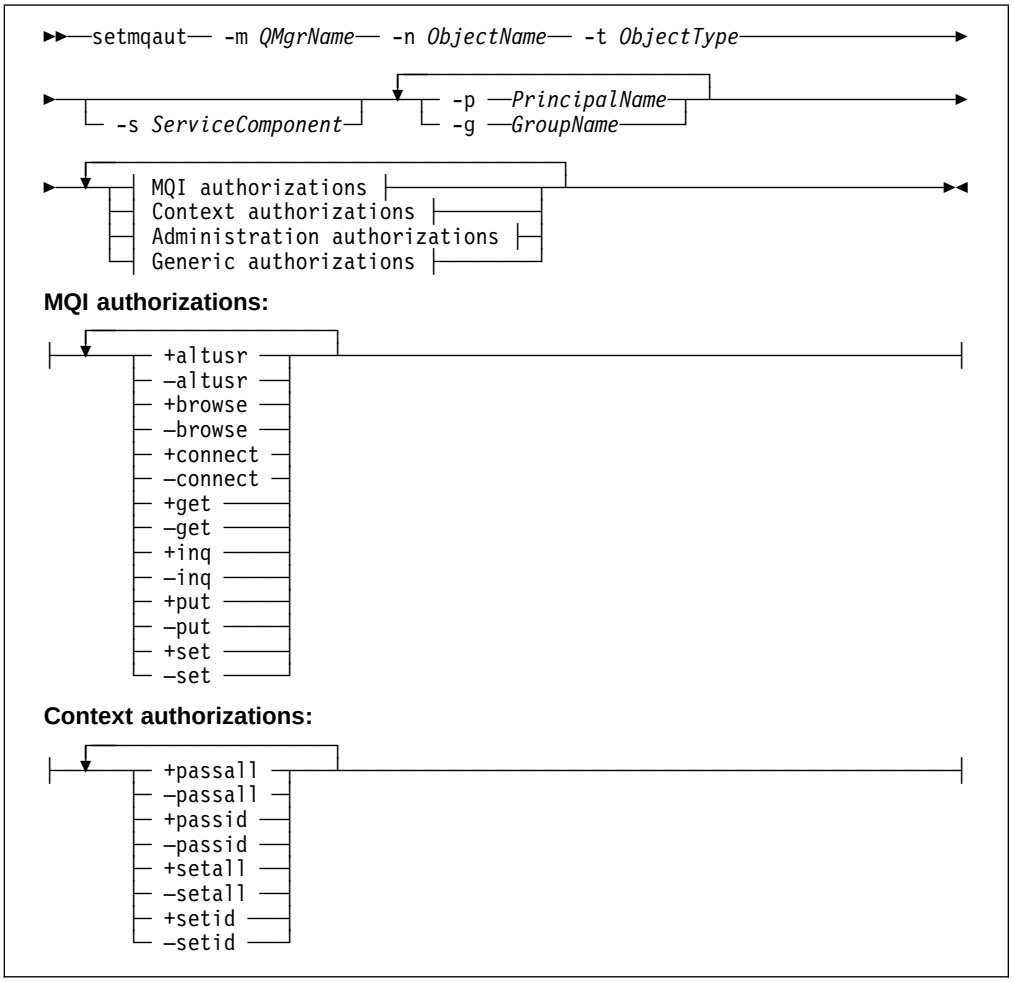
Purpose

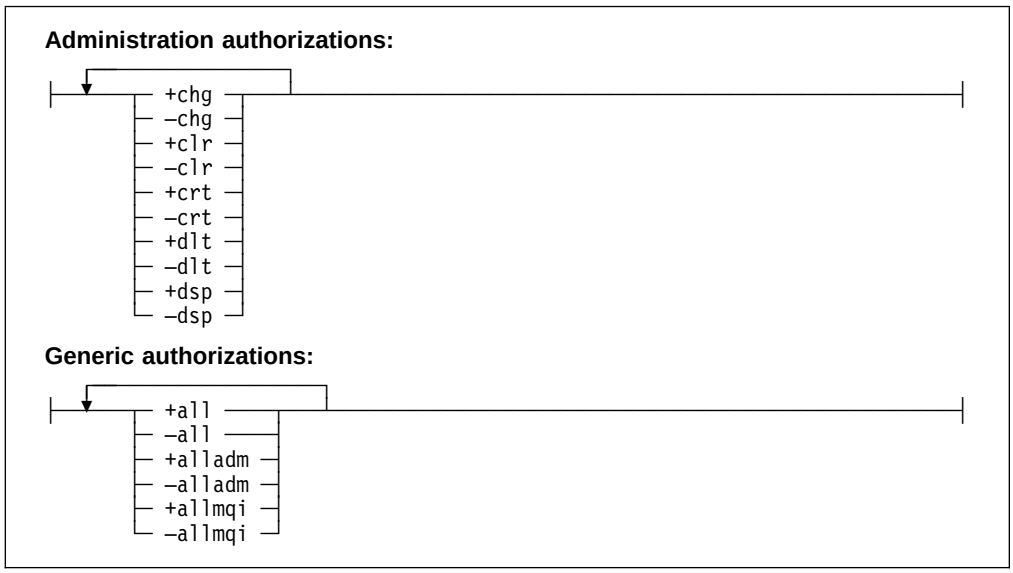
Use the **setmqaut** command to change the authorizations to an object or to a class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

In MQSeries for OS/2 Warp only, you can use this command to specify authorizations only if an authorization service component has been installed for the current queue manager. By default, no such component is supplied with MQSeries for OS/2 Warp.

For more information about authorization service components, see Chapter 12, “Authorization service” in the *MQSeries Programmable System Management* book.

Syntax





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.

Note: In MQSeries for UNIX systems, if you specify a set of authorizations for a principal, the same authorizations are given to all principals in the same primary group.

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by '+' or '–'. For example, if you include `+put` in the authorization list, you are giving authority to issue MQPUT calls against a queue. Alternatively, if you include `–put` in the authorization list, you are removing the authorization to issue MQPUT calls.

Authorizations can be specified in any order provided that they do not clash. For example, specifying `allmqi` with `set` causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

Required parameters

-n *ObjectName*

Specifies the name of the object for which the authorizations are to be changed. You must not use a generic name.

This parameter is optional if you are changing the authorizations of your default queue manager.

-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**
- **nl** or **namelist**

-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.

This parameter is optional if you are changing the authorizations of your default queue manager.

Optional parameters

-p *PrincipalName*

Specifies the name of the principal for which the authorizations are to be changed.

For MQSeries for Windows NT only, the name of the principal can optionally include a domain name which should be specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see “Using the @ symbol in user ID names” on page 106.

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`.

Table 20 shows the authorities that can be given to the different object types.

Table 20. Specifying authorizations for different object types

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

Authorizations for MQI calls

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

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

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

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

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

put — Put a message on a specific queue by issuing an MQPUT call.

set — Set attributes on a queue from the MQI by issuing an MQSET call.

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

Authorizations for context

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

Authorizations for commands

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

Authorizations for generic operations

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

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
```

3. In this example, the authorization list specifies that user group `waltz` has authority to create and delete queue manager `saturn.queue.manager`.

```
setmqaut -m saturn.queue.manager -t qmgr -g waltz +crt +dlt
```

Related commands

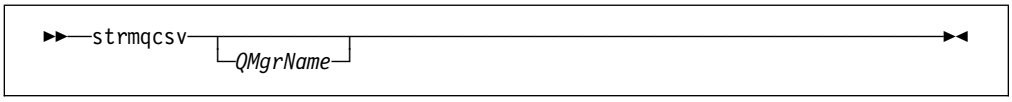
dspmqaut	Display authority

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.

Syntax



Required parameters

None

Optional 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

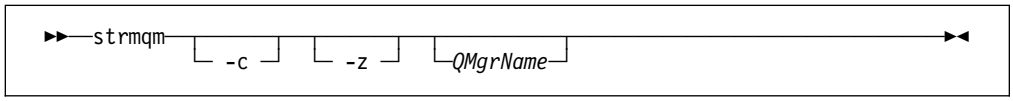
endmqcsv	End a command server
dspmqcsv	Display the status of a command server

strmqm (Start queue manager)

Purpose

Use the **strmqm** command to start a local queue manager.

Syntax



Optional parameters

-c Starts the queue manager, redefines the default and system objects, then stops the queue manager. (The default and system objects for a queue manager are created initially by the **crtmqm** command.) Any existing system and default objects belonging to the queue manager are replaced if you specify this flag.

-z Suppresses error messages.

This flag is used within 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.

QMgrName

Specifies the name of a local queue manager to be started. If omitted, the default queue manager is started.

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)

Special note

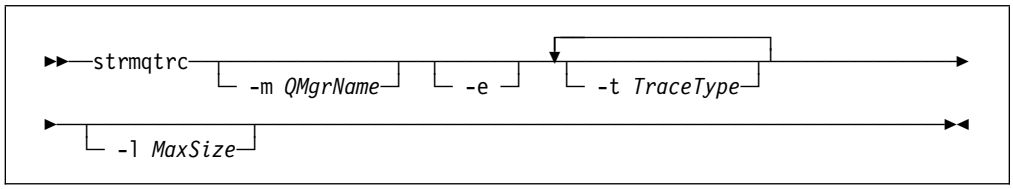
The **strmqtrc** command is not supported by MQSeries for AIX.

Purpose

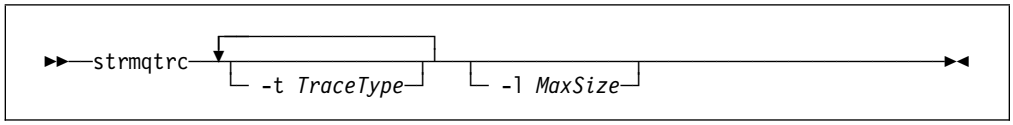
Use the **strmqtrc** command to enable tracing. This command can be run regardless of whether tracing is enabled. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

Syntax

The syntax of this command in MQSeries for HP-UX and Sun Solaris is as follows:



The syntax of this command in MQSeries for OS/2 Warp and Windows NT is as follows:



Description

Different levels of trace detail can be requested. For each flow tracetype value you specify, including -t all, specify either -t parms or -t detail to obtain the appropriate level of trace detail. If you do not specify either -t parms or -t detail for any particular trace type, only a default-detail trace is generated for that trace type.

In MQSeries for HP-UX and Sun Solaris, the output file is always created in the directory /var/mqm/trace.

In MQSeries for OS/2 Warp and Windows NT, the output file is created in the \<mqmwork\errors directory, where <mqmwork> is the directory selected when MQSeries was installed to hold MQSeries data files.

For examples of trace data generated by this command see “Tracing” on page 263.

Optional parameters

-m *QMgrName*

Is the name of the queue manager to be traced.

A queue manager name and the -m flag can be specified on the same command as the -e flag. If more than one trace specification applies to a given entity being traced, the actual trace includes all of the specified options.

It is an error to omit the -m flag and queue manager name, unless the -e flag is specified.

This parameter is not valid in MQSeries for OS/2 Warp and Windows NT

-e If this flag is specified, early tracing is requested. Consequently, it is possible to trace the creation or startup of a queue manager. 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.

This parameter is not valid in MQSeries for OS/2 Warp and Windows NT

-t *TraceType*

Identifies the points to be traced, and specifies the amount of trace detail to be recorded. If this flag is omitted, all trace points are enabled, and a default-detail trace is generated.

Alternatively, one or more of the options in the following list can be supplied.

If multiple trace types are supplied, each ***must*** have its own -t flag. Any number of -t flags can be specified, provided that each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

all Output data for every trace point in the system. This is also the default if the -t flag is not specified. The all parameter activates tracing at default detail level.

api Output data for trace points associated with the MQI and major queue manager components.

commentary
Output data for trace points associated with comments in the MQSeries components.

comms
Output data for trace points associated with data flowing over communications networks.

csdata
Output data for trace points associated with internal data buffers in common services.

csflows
Output data for trace points associated with processing flow in common services.

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

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

lqmflows
Output data for trace points associated with processing flow in the local queue manager.

otherdata
Output data for trace points associated with internal data buffers in other components.

otherflows
Output data for trace points associated with processing flow in other components.

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

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

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

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

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

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

-l `MaxSize`
The value of `MaxSize` denotes the maximum size of a trace file (AMQ`nnnn`.TRC) in millions of bytes. For example, if you specify a MaxSize of 1, the size of the trace is limited to 1 million bytes.

When a trace file reaches the specified maximum, it is renamed from AMQ`nnnn`.TRC to AMQ`nnnn`.TRS and a new AMQ`nnnn`.TRC file is started. All trace files are restarted when the maximum limit is reached. If a previous copy of an AMQ`nnnn`.TRS file exists, it will be deleted.

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 processing flow from common services and the local queue manager for a queue manager called QM1 in MQSeries for UNIX systems. Trace data is generated at the default level of detail.

```
strmqtrc -m QM1 -t csflows -t lqmflows -t parms
```

This command enables high-detail tracing of the processing flow for all components in MQSeries for OS/2 Warp or Windows NT:

```
strmqtrc -t all -t detail
```

Related commands

dspmqtrc	Display formatted trace output
endmqtrc	End MQSeries trace

Part 3. Appendixes

Appendix A. System and default objects

When you create a queue manager using the **crtmqm** control command, the system objects and the default objects are created automatically.

- The system objects are those MQSeries objects required for the operation of a queue manager or channel.
- The default objects define all of the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

The following tables list the system and default objects created by **crtmqm**:

- Table 21 lists the system and default queue objects.
- Table 22 lists the system and default channel objects.
- Table 23 lists the system and default namelist objects.
- Table 24 lists the system and default process objects.

Table 21. System and default objects - queues

Object name	Description
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue which holds the synchronization data for channels.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered message queue).
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
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.

Table 22. System and default objects - channels

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

Table 23. System and default objects - namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 24. System and default objects - processes

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

Windows NT default configuration objects

You set up a default configuration using either the MQSeries First Steps application or the MQSeries Postcard application.

Note: You cannot set up a default configuration if other queue managers exist on your computer.

Furthermore, you cannot set up a default configuration if the TCP/IP address used by the machine is obtained from a Dynamic Host Configuration Protocol (DHCP) server.

Many of the names used for the Windows NT Default Configuration objects involve the use of a fully-qualified TCP/IP name. In all cases where this name has to be truncated, if the last character is a period (.), it is removed.

Any characters within the fully-qualified machine name that are not valid for MQSeries object names (for example, hyphens) are replaced by an underscore character.

Valid characters for MQSeries object names are: a to z, A to Z, 0 to 9 and the four special characters,
/, %, ., and _.

The cluster name for the Windows NT default configuration is the TCP/IP domain-name prefixed with the characters “CL_”. The maximum length of this name is 48 characters. Names exceeding this limit are truncated to 48 characters.

If the queue manager is not a repository queue manager, the objects listed in Table 25 are created.

Table 25. Objects created by the Windows NT Default Configuration application

Object	Name
Queue manager	The fully-qualified machine name prefixed with the characters “QM_”. The maximum length of the fully-qualified machine name is 48 characters. Names exceeding this limit are truncated at 48 characters. The queue manager is created as the default queue manager and is a cluster queue manager in the Default Configuration cluster. The queue manager has a command server, a channel listener, and channel initiator associated with it. The channel listener listens on the standard MQSeries port, that is port number 1414. Any other queue managers created on this machine should not use port 1414 while the default configuration queue manager still exists.
Generic cluster receiver channel	The fully-qualified machine name prefixed with the character “TO_”. The maximum length of this name is 20 characters. Names exceeding this length are truncated at 20 characters.
Cluster sender channel	The fully-qualified machine name of the repository this machine is using, prefixed with the characters “TO_”. The maximum length of this name is 20 characters. Names exceeding this length are truncated at 20 characters.
Local message queue	The local message queue is called “default”.
Local message queue for use by the MQSeries Postcard application	The local message queue for use by the MQSeries Postcard application is called “postcard”.
Server connection channel	The server connection channel allows clients to connect to the queue manager. It is the fully-qualified machine name prefixed with the characters “S_”. The maximum length of this name is 20 characters. Names exceeding this length are truncated at 20 characters.

If the queue manager is a repository queue manager, the default configuration is similar to that described in Table 25, but with the following differences:

- The Queue Manager is defined as a repository queue manager for the default configuration cluster.
- There is no cluster sender channel defined.
- A local cluster queue that is the short name of the machine prefixed with the characters “clq_default_”. The maximum length of this name is 48 characters. Names exceeding this length are truncated at 48 characters.

If you request remote administration facilities, the server connection channel, SYSTEM.ADMIN.SVRCONN is also created.

Appendix B. Directory structure (UNIX systems)

Figure 64 on page 348 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 one of the following:

- "Chapter 3. Installing the MQSeries for AIX Server" in the *MQSeries for AIX V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for HP-UX Server" in the *MQSeries for HP-UX V5.1 Quick Beginnings* book
- "Chapter 3. Installing the MQSeries for Sun Solaris Server" in the *MQSeries for Sun Solaris V5.1 Quick Beginnings* book

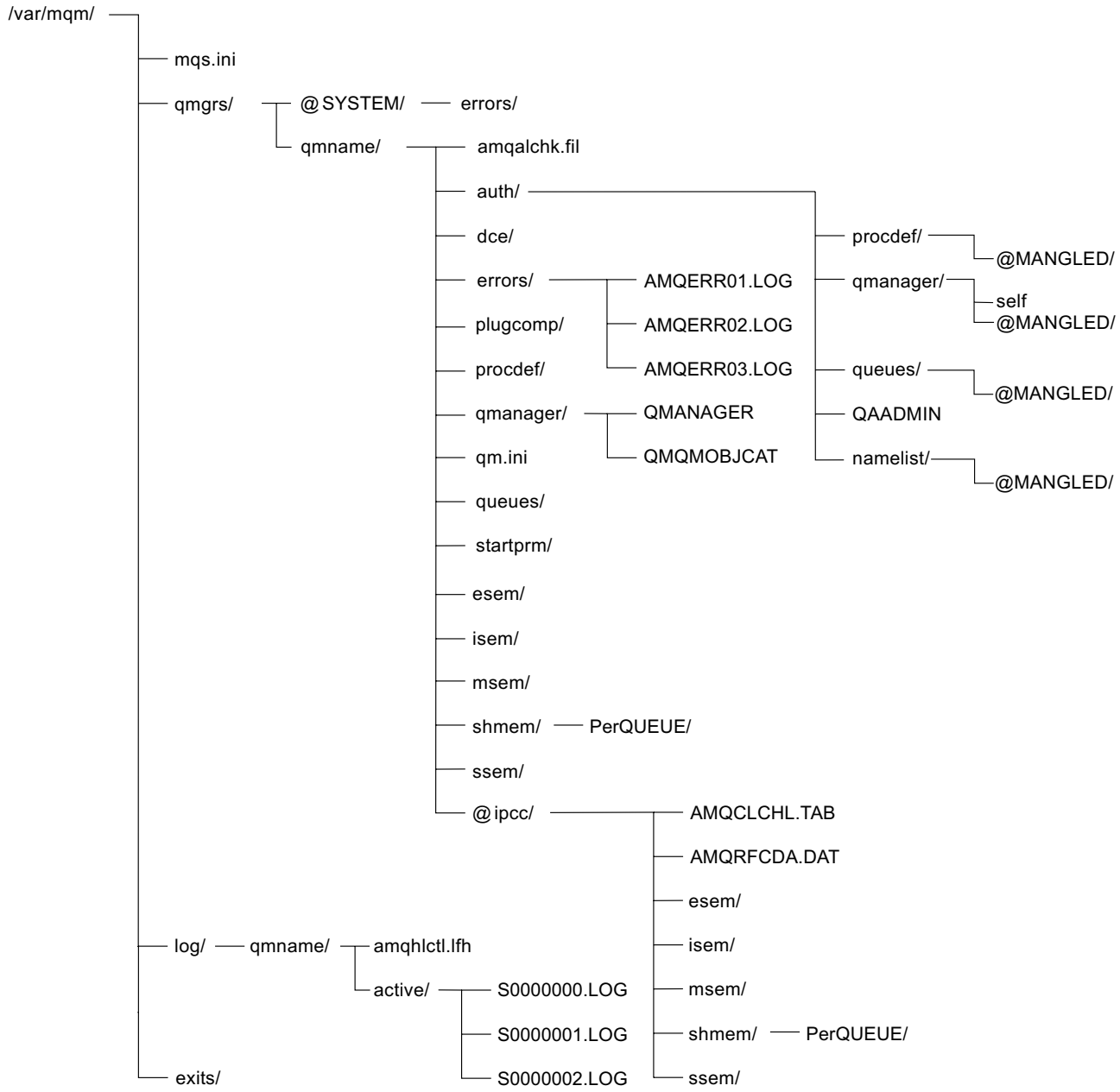


Figure 64. Default directory structure (UNIX systems) after a queue manager has been started

In Figure 64, 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.

@MANGLED
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.

@MANGLED
This file contains the authority stanzas for the process definition class.

qmanager/

@MANGLED
This file contains the authority stanzas for the queue manager class.

self This file contains the authority stanzas for the queue manager object.

queues/ This directory contains a file for each queue. Each file contains the authority stanzas for the associated queue.

@MANGLED
This file contains the authority stanzas for the queue class.

namelist/ This directory contains a file for each namelist. Each file contains the authority stanzas for the associated namelist.

@MANGLED
This file contains the authority stanzas for the namelist.

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” on page 27.

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 “Understanding MQSeries file names” on page 27.

startprm/ Directory containing temporary files used internally.

esem/
isem/
msem/
shmem/ Directories containing files used internally.

- **PerQUEUE/** Directory containing files used internally.

ssem/
@ipcc/ Directory containing files used internally.

- **AMQCLCHL.TAB** Client channel table file.
- **AMQRFCDA.DAT** Channel table file.
- **esem/**
- **isem/**
- **msem/**
- **shmem/** Directories containing files used internally.
  - **PerQUEUE/** Directory containing files used internally.
- **ssem/**

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 S0000000.LOG, S0000001.LOG, S0000002.LOG, and so on.

Appendix C. Directory structure (OS/2)

The following directories and files are found under the root `C:\MQM\QMGRS\QMNAME\`. If you have installed MQSeries for OS/2 Warp under different directories, the root is modified appropriately.

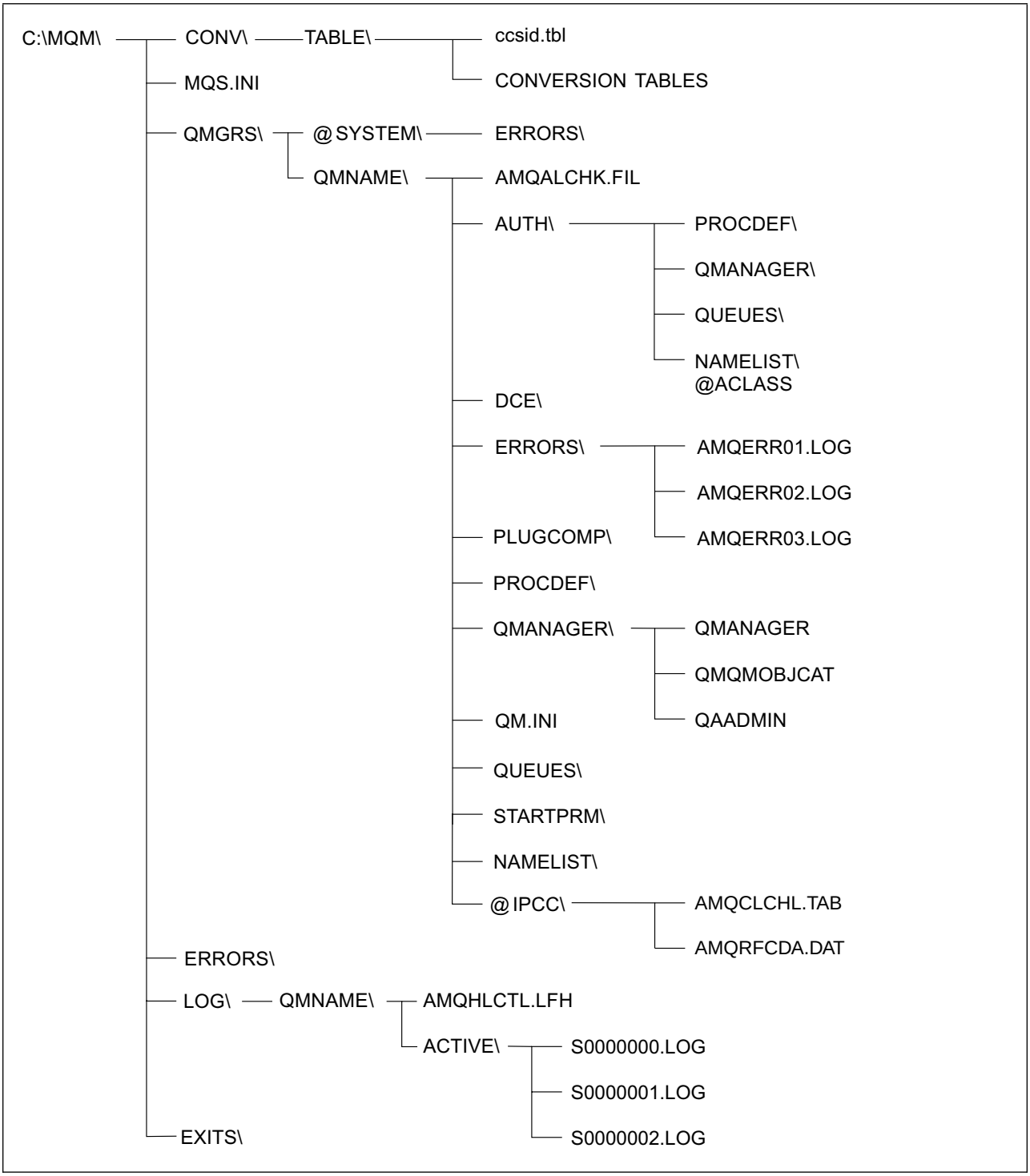


Figure 65. Default file tree (OS/2) after a queue manager has been started. If you are using a FAT system, the name QMQMOBJCAT will be transformed.

Figure 65 shows the general layout of the data and log directories. The layout is representative of MQSeries after a queue manager has been in use for some time. However, the actual structure that you have depends on the operations that have occurred on the queue manager. A brief description of the files follows.

Notes:

1. The directory and file names are all shown in uppercase. The case depends on the file system you are using (FAT or HPFS).
2. The queue manager names may have been transformed. See "Understanding MQSeries file names" on page 27 for more information about name transformation.

AMQALCHK.FIL
Checkpoint file containing information about last checkpoint.

AUTH

PROCDEF\ Empty directory reserved for authority parameters.

QMANAGER\
Empty directory reserved for authority parameters.

QUEUES\ Empty directory reserved for authority parameters.

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 has a file in here.

Where possible, the file name matches the associated process definition name but some characters have to be altered.

There may be a directory called @MANGLED here containing process definitions with transformed or mangled names.

QMANAGER

QMANAGER The queue manager object.

QMQMOBJCAT
The object catalogue containing the list of all MQSeries objects, used internally.

Note: If you are using a FAT system, this name will be transformed and a subdirectory created containing the file with its name transformed.

QAADMIN File used internally for controlling authorizations.

QM.INI Queue manager configuration file

QUEUES Each queue has a directory here containing a single file called Q.

Where possible, the directory name matches the associated queue name but some characters have to be altered.

There may be a directory called @MANGLED here containing queues with transformed or mangled names.

STARTPRM Directory containing temporary files used internally.

@IPCC

AMQCLCHL.TAB
: File containing the client channel table

AMQRFCDA.DAT
: File containing the channel table

AMQRSYNA.DAT
: Channel synchronization file

Queue manager log directory structure

The following directories and files are found under `C:\MQM\LOG\QMNAME\`. If you have installed the product under different directories or specified different log paths in the configuration file, the root will be modified appropriately.

The following subdirectories and files will 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 S0000000.LOG, S0000001.LOG, S00000002.LOG, and so on.

Appendix D. Directory structure (Windows NT)

Figure 66 shows some of the directories and files found under the root `c:\mqm\`. If you have installed MQSeries for Windows NT under different directories, the root is modified appropriately.

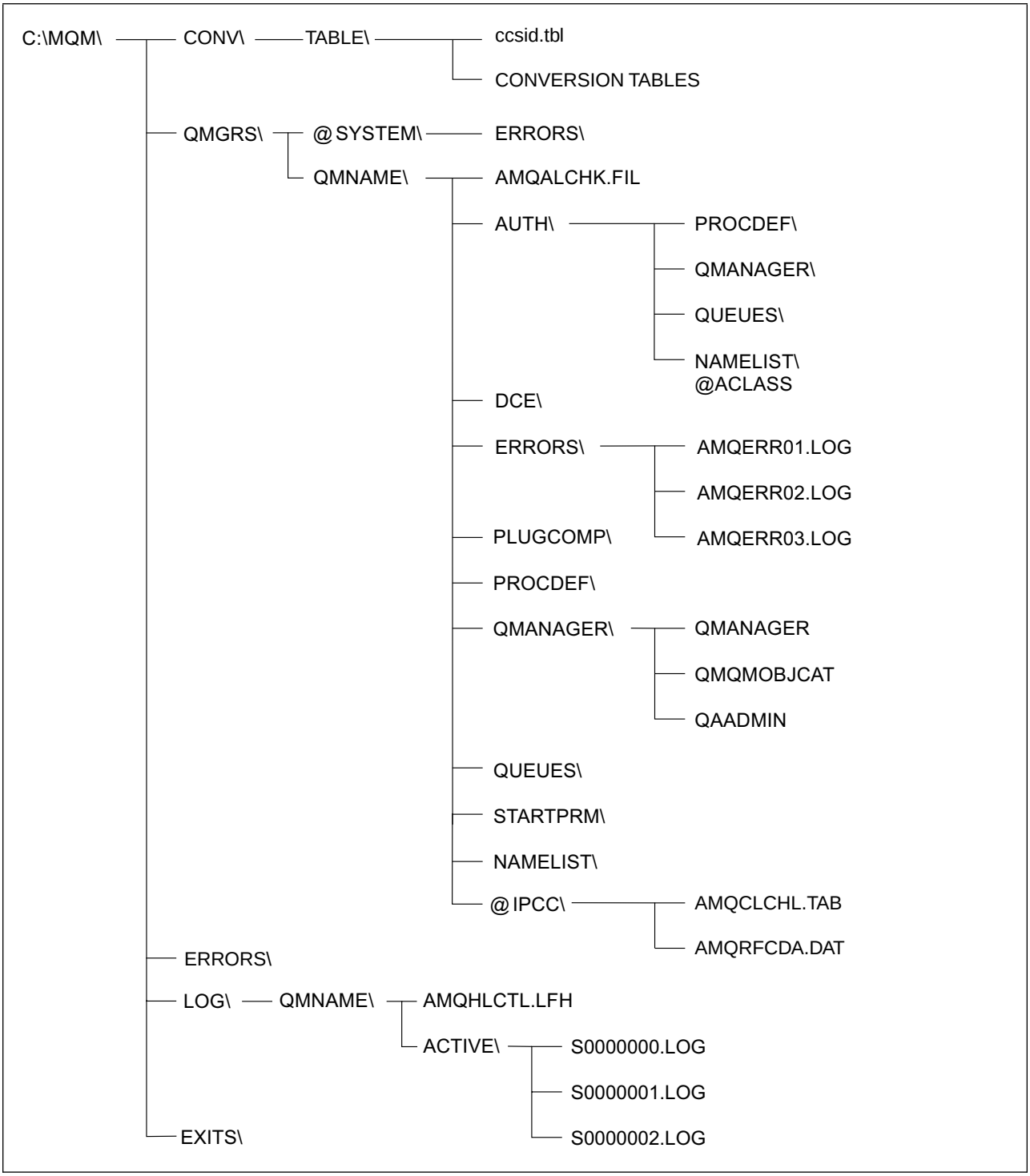


Figure 66. Default file tree (Windows NT) after a queue manager has been started. If you are using a FAT system, the name QMQMOBJCAT is transformed.

Figure 66 shows the general layout of the data and log directories. The layout is representative of MQSeries after a queue manager has been in use for some time. However, the actual structure that you have depends on the operations that have occurred on the queue manager. A brief description of the files follows.

Notes:

1. The directory and file names are all shown in upper case. The case depends on the file system you are using (NTFS, HPFS, or FAT).
2. The queue manager names may have been transformed. See "Understanding MQSeries file names" on page 27 for more information about name transformation.

AMQALCHK.FIL
Checkpoint file containing information about last checkpoint.

AUTH

NAMELIST Directory reserved for authority parameters.

PROCDEF\ Directory reserved for authority parameters.

QMANAGER\
Directory reserved for authority parameters.

QUEUES\ Directory reserved for authority parameters.

DCE Empty directory reserved for use by DCE support.

ERRORS The operator message files, from newest to oldest:

AMQERR01.LOG
AMQERR02.LOG
AMQERR03.LOG

NAMELIST Each MQSeries namelist has a file in here.

PLUGCOMP
Empty directory reserved for use by installable services.

PROCDEF Each MQSeries process definition has a file in here.

Where possible, the file name matches the associated process definition name but some characters have to be altered.

There may be a directory called @MANGLED here containing process definitions with transformed or mangled names.

QMANAGER

QMANAGER
The queue manager object.

QMQMOBJCAT
The object catalogue containing the list of all MQSeries objects, used internally.

Note: If you are using a FAT system, this name will be transformed and a subdirectory created containing the file with its name transformed.

QAADMIN File used internally for controlling authorizations.

QUEUES Each queue has a directory here containing a single file called Q.

Where possible, the directory name matches the associated queue name but some characters have to be altered.

There may be a directory called @MANGLED here containing queues with transformed or mangled names.

STARTPRM
Directory containing temporary files used internally.

@IPCC

AMQCLCHL.TAB
File containing the client channel table.

AMQRFCDA.DAT
File containing the channel table.

Queue manager log directory structure

The following directories and files are found under C:\MQM\LOG\QMNAME\. If you have installed the product under different directories or specified different log paths in the configuration file, the root will be modified appropriately.

The following subdirectories and files will 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 S0000000.LOG, S0000001.LOG, S00000002.LOG, and so on.

Appendix E. Stopping and removing queue managers manually

If the standard methods for stopping and removing queue managers fail, you can resort to the more drastic methods described here.

Stopping a queue manager manually

The standard way of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, you can use one of the procedures described here to stop it manually.

Stopping queue managers in MQSeries for UNIX systems

To stop a queue manager running under MQSeries for UNIX systems:

1. Find the process IDs of the queue manager programs that are still running using the **ps** command. For example, if the queue manager is called `QMNAME`, the following command can be used:

   ```
   ps -ef | grep QMNAME
   ```

2. End any queue manager processes that are still running. Use the **kill** command, specifying the process IDs discovered using the **ps** command.

   Note: Processes that fail to stop can be ended using **kill -9**.

   End the processes in the following order:

amqpcsea	command server
amqhasmx	logger
amqharmx	log formatter (LINEAR logs only)
amqzllp0	checkpoint processor
amqzlaa0	queue manager agents
amqzxma0	processing controller
amqrrmfa	Repository process (for clusters)

Note: Manual stopping of the queue manager may result in FFSTs 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 having been stopped using this method.

Attention!

If you do not shut down a queue manager properly, you run the risk of MQSeries not tidying up operating system resources such as semaphores and shared memory sets. This can result in a gradual degradation of system performance and in you having to reboot your system.

Stopping queue managers in MQSeries for Windows NT

To stop a queue manager running under MQSeries for Windows NT:

1. List the names (IDs) of the processes currently running using the Windows NT Process Viewer (PView)
2. Stop the processes using PView in the following order (if they are running):

AMQPCSEA.EXE	The command server
AMQHASMN.EXE	The logger
AMQHARMN.EXE	Log formatter (LINEAR logs only)
AMQZLLP0.EXE	Checkpoint process
AMQZLAA0.EXE	LQM agents
AMQZTRCN.EXE	Trace
AMQZXMA0.EXE	Execution controller
AMQXSSVN.EXE	Shared memory servers
AMQRRMFA.EXE	The repository process (for clusters)

3. Stop the queue manager service using the Windows NT Control Panel.
4. If you have tried all methods and the queue manager has not stopped, reboot your system.

Stopping queue managers in MQSeries for OS/2 Warp

To stop a queue manager running under MQSeries for OS/2 Warp:

1. If you have access to an appropriate utility (equivalent to the UNIX **ps** command), list the names (IDs) of the processes currently running.
2. If you have access to a utility that stops processes (equivalent to the UNIX **kill** command), stop them in the following order:

AMQPCSEA.EXE	The command server
AMQHASM2.EXE	The logger
AMQHARM2.EXE	Log formatter (LINEAR logs only)
AMQZLLP0.EXE	Checkpoint process
AMQZLAA0.EXE	LQM agents
AMQZXMA0.EXE	Execution controller
AMQXSSV2.EXE	Shared memory servers
AMQRRMFA.EXE	The repository process (for clusters)

Note: If you do not have access to a suitable utility, and you have tried all other methods, you must reboot your system.

Removing queue managers manually

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 processes described here can be used.

Removing queue managers in MQSeries for UNIX systems

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. This is because 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.

Removing queue managers in MQSeries for Windows NT

If you encounter problems with the dltmqm command in MQSeries for Windows NT, use the following procedure to delete a queue manager:

1. Type REGEDT32 from the command prompt to start the Registry Editor.
2. Select the HKEY_LOCAL_MACHINE window.
3. Navigate the tree structure in the left-hand pane of the Registry Editor to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion
   ```

   Make a note of the values within this key called WorkPath and LogPath. Within each of the directories named by these values, you are going to delete a subdirectory containing the data for the queue manager which you are trying to delete. You now need to find out the name of the subdirectory which corresponds to your queue manager.
4. Navigate the tree structure to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
   Configuration\QueueManager
   ```

   Within this key there is a key for each of the queue managers on this computer containing the configuration information for the queue manager. The name of this queue manager key is the name of the subdirectory in which the queue manager's data is stored in the file system. This is not necessarily identical to the name of the queue manager. By default, this name is the same as the

queue manager name, but the name may be a transformation of the queue manager name.

5. Examine the keys within the current key. Look for the key which contains a value called `Name`. `Name` contains the name of the queue manager you are trying to delete. Make a note of the name of the key containing the name of the queue manager you are trying to delete. This is the subdirectory name.
6. Locate the queue manager data directory. The name of this directory is the WorkPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
7. Locate the queue manager's log directory. The name of this directory is the LogPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
8. Remove the registry entries which refer to the deleted queue manager. First, navigate the tree structure in the Registry Editor to the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
   Configuration\DefaultQueueManager
   ```

9. If the value called Name within this key matches the name of the queue manager you are deleting, delete the DefaultQueueManager key.
10. Navigate the tree to the following key:

    ```
    HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\Services
    ```

11. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.
12. Navigate the tree to the following key:

    ```
    HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\
    Configuration\QueueManager
    ```

13. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.

Removing queue managers from the automatic start-up list

If for any reason the MQSeries Services snap-in cannot be used to change the startup state of a particular queue manager, use the following routine to carry out the same procedure manually:

1. Stop the MQSeries Services snap-in either from the task bar icon or from the control panel.
2. Type REGEDT32 on the command line.
3. Select the HKEY_LOCAL_MACHINE window.
4. Navigate the tree structure to find the following key:

   ```
   LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\Services\<QMgrName>\QueueManager
   ```

5. Change the startup value to zero. (1 means automatic and 0 means manual.)
6. Close the registry editor.

Removing queue managers in MQSeries for OS/2 Warp

If you encounter problems with the **dltmqm** command in MQSeries for OS/2 Warp, use the following procedure to delete a queue manager:

1. Locate the queue manager directory from the `mqs.ini` configuration file. By default, this location is:

   ```
   C:\MQM\QMGRS\<QMgrName>
   ```

   where `<QMgrName>` (or its transformed equivalent) is the name of the queue manager to be deleted.
2. Delete this directory, all subdirectories and files.
3. Locate the associated log directory from the mqs.ini file.
4. Delete the directory, all subdirectories and files.
5. Remove its `QueueManager` stanza from `mqs.ini`.
6. Remove the `DefaultQueueManager` stanza, if the queue manager being deleted is the default queue manager.

Appendix F. User identifier service

This information applies to MQSeries for OS/2 Warp only.

By default, the user ID associated with applications running under OS/2 is `os2`. The queue manager inserts this user ID into the context fields of any messages that are sent by the application.

The user identifier service enables a user-defined user ID to be substituted for the default. When the user identifier service is active, this user ID is accessed by the local queue manager when an application issues an MQCONN request. Thereafter, the queue manager inserts the new user ID in the context field of any message sent by the application.

The mechanism for defining the new user ID must be user-written. For example, in the sample AMQSZFC0, which is supplied with MQSeries for OS/2 Warp, the user ID is defined in an OS/2 environment variable. Once you have added the appropriate stanza to the queue manager configuration file, you simply type this in at the keyboard. To get the queue manager to recognize the new user ID, you must restart the queue manager. The user identifier service is described in detail in Chapter 14, “User identifier service” in the *MQSeries Programmable System Management* book.

Appendix G. Comparing command sets

The tables in this appendix compare the facilities available from the different administration command sets, and state, for MQSeries for Windows NT only, whether you can perform each function from within the MQSeries Explorer snap-in and the MQSeries Services snap-in.

Only those MQSC commands that are supported by MQSeries Version 5.1 are shown.

MQSC commands can be issued in the form of MQSC scripts from MQSeries Web Administration. See Chapter 5, “Using MQSeries Web Administration” on page 43 for more information.

Commands for queue manager administration

Table 26. Commands for queue manager administration

PCF commands	MQSC commands	Control commands	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Change Queue Manager	ALTER QMGR	No equivalent	Yes	No
(Create queue manager) [1]	No equivalent	**crtmqm**	Yes	Yes
(Delete queue manager) [1]	No equivalent	**dltmqm**	Yes	Yes
Inquire Queue Manager	DISPLAY QMGR	No equivalent	Yes	No
(Stop queue manager) [1]	No equivalent	**endmqm**	Yes	Yes
Ping Queue Manager	PING QMGR	No equivalent	No	No
(Start queue manager) [1]	No equivalent	**strmqm**	Yes	Yes

Notes:

1. Not available as PCF commands
2. Part of the MQSeries Services snap-in

Commands for command server administration

Table 27. Commands for command server administration

Description	PCF command	MQSC command	Control command	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Display command server	No equivalent	No equivalent	**dspmqcsv**	No	Yes
Start command server	No equivalent	No equivalent	**strmqcsv**	No	Yes
Stop command server	No equivalent	No equivalent	**endmqcsv**	No	Yes

Commands for queue administration

Table 28. Commands for queue administration

PCF command	MQSC command	Control command	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE	No equivalent	Yes	No
Clear Queue	CLEAR QUEUE	No equivalent	Yes	No
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	No equivalent	No	No
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE	No equivalent	Yes	No
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE	No equivalent	Yes	No
Inquire Queue	DISPLAY QUEUE	No equivalent	Yes	No
Inquire Queue Names	DISPLAY QUEUE	No equivalent	Yes	No

Commands for process administration

Table 29. Commands for process administration

PCF command	MQSC command	Control command	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Change Process	ALTER PROCESS	No equivalent	Yes	No
Copy Process	DEFINE PROCESS(x) LIKE(y)	No equivalent	No	No
Create Process	DEFINE PROCESS	No equivalent	Yes	No
Delete Process	DELETE PROCESS	No equivalent	Yes	No
Inquire Process	DISPLAY PROCESS	No equivalent	Yes	No
Inquire Process Names	DISPLAY PROCESS	No equivalent	Yes	No

Commands for channel administration

Table 30. Commands for channel administration

PCF command	MQSC command	Control command	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Change Channel	ALTER CHANNEL	No equivalent	Yes	No
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	No equivalent	No	No
Create Channel	DEFINE CHANNEL	No equivalent	Yes	No
Delete Channel	DELETE CHANNEL	No equivalent	Yes	No
End Listener	No equivalent	ENDMQLSR	No	Yes
Inquire Channel	DISPLAY CHANNEL	No equivalent	Yes	No
Inquire Channel Names	DISPLAY CHANNEL	No equivalent	Yes	No
Ping Channel	PING CHANNEL	No equivalent	Yes	No
Reset Channel	RESET CHANNEL	No equivalent	Yes	No
Resolve Channel	RESOLVE CHANNEL	No equivalent	Yes	No
Start Channel	START CHANNEL	**runmqchl**	Yes	Yes
Start Channel Initiator	START CHINIT	**runmqchi**	No	Yes
Start Channel Listener	START LISTENER	**runmqlsr**	No	Yes
Stop Channel	STOP CHANNEL	No equivalent	Yes	Yes

Other control commands

Table 31. Other control commands

Description	PCF command	MQSC command	Control command	MQSeries Explorer equivalent?	MQSeries Services snap-in equivalent?
Create MQSeries conversion exit	No equivalent	No equivalent	**crtmqcvx**	No	No
Dump MQSeries log	No equivalent	No equivalent	**dmpmqlog**	No	No
Display authority	No equivalent	No equivalent	**dspmqaut**	No	No
Display files used by objects	No equivalent	No equivalent	**dspmqfls**	No	No
Display MQSeries formatted trace	No equivalent	No equivalent	**dspmqtrc** [2]	No	No
Display MQSeries transactions	No equivalent	No equivalent	**dspmqtrn**	No	No
End MQSeries trace	No equivalent	No equivalent	**endmqtrc** [1]	No	Yes
Record media image	No equivalent	No equivalent	**rcdmqimg**	No	No
Recreate media object	No equivalent	No equivalent	**rcrmqobj**	No	No
Resolve MQSeries transactions	No equivalent	No equivalent	**rsvmqtrn**	No	No
Run dead-letter queue handler	No equivalent	No equivalent	**runmqdlq**	No	No
Run MQSC commands	No equivalent	No equivalent	**runmqsc**	No	No
Run trigger monitor	No equivalent	No equivalent	**runmqtrm**	No	Yes
Run client trigger monitor	No equivalent	No equivalent	**runmqtmc**	No	No
Set or reset authority	No equivalent	No equivalent	**setmqaut**	No	No
Start MQSeries trace	No equivalent	No equivalent	**strmqtrc** [1]	No	Yes

Notes:

1. Not supported by MQSeries for AIX.
2. Supported by MQSeries for HP-UX and MQSeries for Sun Solaris only.

Appendix H. Using the User Datagram Protocol

MQSeries for AIX V5.1 supports the User Datagram Protocol (UDP), a part of the Internet suite of protocols, as an alternative to TCP. You might decide to use UDP instead of TCP for your mobile radio network, where you need to reduce the traffic, and therefore the cost, on a packet radio data network.

UDP uses the Internet Protocol (IP) to deliver datagrams, the basic unit of information for UDP.

You can use UDP to send message data between MQSeries for Windows Version 2.02 systems (that is, with CSD 2 installed) and MQSeries for AIX Version 5.1 server systems.

Note: UDP is supplied as part of the operating system or TCP/IP suite you are using; you do not need to buy and install a separate UDP product.

This appendix describes how to:

- Configure MQSeries to use UDP, by using MQSC commands
- Optionally, write an exit program to monitor when data can be sent

It also provides some hints and tips on what to do to tailor the UDP support in MQSeries to suit your needs.

Configuring MQSeries for UDP

You configure MQSeries to use UDP by using MQSC commands.

You must make sure there is a listener started for UDP by issuing the following command:

```
runmqlsr -m QMgrName -t UDP
```

Notes:

1. You cannot start a listener on AIX by using the START LISTENER MQSC command.
2. Using the **runmqlsr** command means that you must **not** add entries in the /etc/services and /etc/inetd.conf files for UDP on MQSeries for AIX.

Examples of MQSC command files

The following figures show simple MQSC command files, supplied with MQSeries, with the channel transport type (the TRPTYPE parameter) set to UDP:

- Figure 67 on page 374 shows the file EARTH.TST
- Figure 68 on page 375 shows the file MOON.TST

Both of these files are supplied in directory \mqw\samples. They define the queues and channels you use if you follow the procedure for setting up and verifying two queue managers.

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager.
DEFINE QLOCAL('SAMPLE.EARTH.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)

* Define the remote queue.
* The sample application should put messages on this queue.
DEFINE QREMOTE('SAMPLE.EARTH.REMOTE') REPLACE  +
       DESCR('Remote queue defined on EARTH') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.MOON.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('MOON') +
* This is the name local transmission queue to be used
       XMITQ('SAMPLE.EARTH.XMIT')

* Define the channel that will remove messages from the transmission
* queue SAMPLE.EARTH.XMIT and send them to the machine specified
* by CONNAME.
*
* Change CONNAME to the IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('EARTH.TO.MOON')  CHLTYPE(SDR) TRPTYPE(UDP) +
       XMITQ('SAMPLE.EARTH.XMIT') +
       CONNAME('MOON IP machine name') +
       DESCR('Sender channel for messages to queue manager MOON') +
       REPLACE

* Define the channel that will accept messages from the remote
* queue manager on the machine specified by CONNAME.
*
* Change CONNAME to the IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('MOON.TO.EARTH') CHLTYPE(RQSTR) TRPTYPE(UDP) +
       CONNAME('MOON IP machine name') +
       DESCR('Requester channel for messages from queue manager MOON') +
       REPLACE

* Define the local queue where the remote machine will place
* its messages.
* The sample application should get messages from this queue.
DEFINE QLOCAL('SAMPLE.EARTH.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 67. The supplied file EARTH.TST, UDP support

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager
DEFINE QLOCAL('SAMPLE.MOON.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)


* Define the remote queue.
* The sample application should put messages on this queue
DEFINE QREMOTE('SAMPLE.MOON.REMOTE') REPLACE  +
       DESCR('Remote queue defined on MOON') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.EARTH.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('EARTH') +
* This is the name local transmission queue to be used
       XMITQ('SAMPLE.MOON.XMIT')


* Define the channel that will remove messages from the transmission
* queue SAMPLE.MOON.XMIT and send them to the remote queue
* manager.
DEFINE CHANNEL ('MOON.TO.EARTH') CHLTYPE(SVR) TRPTYPE(UDP) +
       XMITQ('SAMPLE.MOON.XMIT') +
       DESCR('Server channel for messages to queue manager EARTH') +
       REPLACE


* Define the channel that will accept messages from the remote
* queue manager.
DEFINE CHANNEL ('EARTH.TO.MOON')  CHLTYPE(RCVR) TRPTYPE(UDP) +
       DESCR('Receiver channel for messages from queue manager EARTH') +
       REPLACE


* Define the local queue where the remote machine will place
* its messages.
* The sample application should get messages from this queue
DEFINE QLOCAL('SAMPLE.MOON.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 68. The supplied file MOON.TST, UDP support

The retry exit

MQSeries allows you to write a C language retry exit. The exit allows your application to suspend data being sent on a channel when communication is not possible (for example, when the mobile user is traveling through a tunnel or is temporarily out of range of a transmitter).

The retry exit can be associated with a monitor program that can assess whether the IP connection is available for sending data. The exit has to be built into an AIX library (in the same way as any other MQSeries library).

The exit is normally called before a datagram is about to be sent but is also called to provide other useful signals.

The retry exit is called under five different conditions:

- When the MQ channel is first initialized; the ExitReason variable is set to a value of MQXR_INIT.
- When the MQ channel is shut down; the ExitReason variable is set to a value of MQXR_TERM.
- Before each datagram is sent; the ExitReason variable is set to a value of MQXR_RETRY.
- When the end of a batch of messages occurs; the ExitReason variable is set to a value of MQXR_END_BATCH.

  Strictly speaking, the UDP transport layer knows nothing about end of batches because this is a concept known only at the queue manager level; however, it happens that at this point the transport layer moves from a series of ccxSend() verbs to a single ccxReceive() verb and back again. This change of mode, from ccxSend() to ccxReceive() is detected and the transport exit is called accordingly.
- When an “information” datagram is received from the remote end of the link; the ExitReason variable is set to a value of MQXR_ACK_RECEIVED.

The following table provides an explanation of the variables.

Variable	Explanation
ExitReason	Reason for invoking the exit (for example, MQXR_RETRY).
ExitUserArea	Exit user area. When the exit is first invoked, the user can return a value here. This value will be presented in this field for all subsequent invocations of the exit for that channel.
TransportType	Transport type. This always has a value of MQXPT_UDP.
RetryCount	The number of times the data has been retried (zero on first entry to exit).
DataLength	Length of data to be sent (in bytes).
SessionId	Session identifier. This is unique for each channel.
GroupId	Group identifier. Identifies the batch of datagrams currently being sent.
DataId	Data identifier. This is an identifier for each datagram.
ExitResponse	Response from exit. The user fills this in on return with a value (for example, MQXCC_OK).
Feedback	Reserved.

If you want to postpone sending a datagram in response to an ExitReason of MQXR_RETRY, you need to block returning from the exit until it is safe to send the datagram. In all other cases, the return from the exit should be immediate.

There are three possible return codes that can be set when returning from the exit:

- MQXCC_OK — this is the normal response.
- MQXCC_CLOSE_CHANNEL — in response to a ExitReason of MQXR_RETRY, this will cause the channel to be closed.
- MQXCC_REQUEST_ACK — in response to a ExitReason of MQXR_RETRY, this will cause the datagram about to be sent to be modified so that it requests the remote end of the link to send an “information” datagram back to indicate that the node can be reached. If this datagram arrives the exit will be invoked again with an ExitReason of MQXR_ACK_RECEIVED.

  Note: If the datagram fails to arrive at the remote node, for any reason, the user will have to repeat the request on the next datagram that is sent.

Other information is available to the user when the exit is called (see the file CMQXC.H for full details). An example called REXIT is supplied (see various files called REXIT.*).

The retry exit name can be defined by the user, who can also change the name of the library that contains the exit. The library should reside in the same a directory as other MQSeries exits. See Chapter 35, “Channel-exit programs” in the *MQSeries Intercommunication* book for general information about MQSeries exits.

You configure the retry exit by editing the qm.ini file.

Hints and tips

Depending on your circumstances, you might need to do one or both of the following to tailor the UDP support in MQSeries to suit your own needs:

- Edit the qm.ini configuration file.

  A sample qm.ini configuration file is shipped in the mqw\qmgrs\ directory. To use it, copy it to the sub-directory for the queue manager and edit it as required.

  The parameters and values in the sample configuration file are:

  ```
  UDP:
  ACKREQ_TIMEOUT = 5
  ACKREQ_RETRY = 60
  CONNECT_TIMEOUT = 5
  CONNECT_RETRY = 60
  ACCEPT_TIMEOUT = 5
  ACCEPT_RETRY = 60
  DROP_PACKETS = 0
  BUNCH_SIZE = 8
  PACKET_SIZE = 2048
  PSEUDO_ACK  = YES
  TRANSPORT:
  RETRY_EXIT = exitname
  ```

  See “The UDP stanza” on page 147 and See “The Transport stanza” on page 149 for descriptions of each of these attributes and how to code them.

- Keep messages to less than 4 MB.

Appendix I. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Trademarks

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

AIX	AS/400	BookManager
CICS	DB2	DB2 Universal Database
FFST	FFST/2	First Failure Support Technology
IBM	IBMLink	IMS
MQ	MQSeries	OS/2
OS/400	OS/390	TXSeries
VisualAge	VSE/ESA	

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

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

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

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

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

Part 4. Glossary and index

Glossary of terms and abbreviations

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

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

A

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

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

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

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

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

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

APAR. Authorized program analysis report.

APPC. Advanced Program-to-Program Communication.

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

application queue. A queue used by an application.

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

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

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

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

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

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

B

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

bag. See *data bag*.

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

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

C

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

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

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

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

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

checkpoint. (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.

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

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

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

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

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

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

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

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

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

command processor. The MQSeries component that processes commands.

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

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

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

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

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

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

context. Information about the origin of a message.

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

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

controlled shutdown. See *quiesced shutdown*.

D

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

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

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

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

DCE. Distributed Computing Environment.

DCI. Data conversion interface.

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

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

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

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

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

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

DLQ. Dead-letter queue.

DQM. Distributed queue management.

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

E

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

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

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

event log. See *application log*.

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

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

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

F

FFST. First Failure Support Technology.

FIFO. First-in-first-out.

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

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

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

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

G

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

H

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

I

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

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

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

IP. Internet Protocol.

L

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

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

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

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

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

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

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

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

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

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

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

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

M

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. (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 at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

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

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

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

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

message queue. Synonym for *queue*.

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

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

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

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

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

MQAI. MQSeries Administration Interface.

MQI. Message queue interface.

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

MQSC. MQSeries commands.

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

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

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

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

N

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

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

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

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

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

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

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

NSI. Name service interface.

NTFS. New Technology File System.

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

O

OAM. Object authority manager.

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

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

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

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

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

P

PCF. Programmable command format.

PCF command. See *programmable command format*.

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

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

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

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

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

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

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

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

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

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

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

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

- User administration applications, to put PCF commands onto the system command input queue of a specified queue manager

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

Contrast with *MQSC*.

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

PTF. Program temporary fix.

Q

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

queue manager. (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, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queuing. See *message queuing*.

quiesced shutdown. (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.

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

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

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

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

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

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

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

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

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

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

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

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The

requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

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

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

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

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

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

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

rollback. Synonym for *back out*.

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

S

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

SEI. Security enabling interface.

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

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

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

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

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

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

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

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

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

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

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

SNA. Systems Network Architecture.

SPX. Sequenced Packet Exchange transmission protocol.

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

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

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

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

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

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

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

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

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

T

TCP. Transmission Control Protocol.

TCP/IP. Transmission Control Protocol/Internet Protocol.

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

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

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

tranid. See *transaction identifier*.

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

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

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

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

transmission program. See *message channel agent*.

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

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

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

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

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

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

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

U

UDP. User Datagram Protocol.

UIS. User identifier service.

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

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be

made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

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

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

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

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

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

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

X

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

Index

Special Characters

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Sending your comments to IBM

MQSeries®

System Administration

SC33-1873-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink®: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MQSeries®

System Administration

SC33-1873-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____

Company or Organization _____

Address _____

_____

EMAIL _____

Telephone _____

Cut along this line

4 Fasten here with adhesive tape

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1873-01