



MQSeries for Digital OpenVMS

GC33-1791-00

System Management Guide

Version 2 Release 2



MQSeries for Digital OpenVMS

GC33-1791-00

System Management Guide

Version 2 Release 2

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix O, "Notices" on page 345.

First edition (May 1997)

This edition applies to the following product:

- MQSeries for Digital OpenVMS Version 2 Release 2

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.

© **Copyright International Business Machines Corporation 1995, 1997. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	xi
Who this book is for	xi
What you need to know to understand this book	xi
How to use this book	xi
MQSeries publications	xii
Evaluating products	xii
Planning	xii
Administration	xii
Application programming	xiii
Problem determination	xiii
Special topics	xiv
Other MQSeries Version 1 publications	xiv
Information about MQSeries on the Internet	xiv

Part 1. Guidance	1
Chapter 1. Introduction	3
MQSeries and message queuing	3
Messages and queues	4
Objects	5
System default objects	10
Administration	10
Clients and servers	11
Extending queue manager facilities	11
Security	12
Chapter 2. Installing MQSeries for Digital OpenVMS	13
Components you can install	13
Installation requirements	14
Installation on Digital OpenVMS Version 6.2	15
Post-installation tasks	19
System configuration	21
Directories that exist after installation	21
Translated messages	22
Verifying your installation	22
Installing clients	26
Chapter 3. Customizing your system	27
Things you can customize	28

Chapter 4. Understanding administration command sets	33
Control commands	33
MQSeries commands (MQSC)	34
PCF commands	35
Comparing command sets	36
Chapter 5. Managing queue managers	39
Getting started	39
Guidelines for creating queue managers	39
Understanding MQSeries file names	42
Working with queue managers	43
Managing the command server for remote administration	47
Chapter 6. Administering local MQSeries objects	49
Supporting application programs that use the MQI	49
Issuing MQSC commands for administration	50
Running MQSC commands from text files	53
If you have problems with MQSC	56
Working with local queues	58
Working with alias queues	65
Working with model queues	67
Managing objects for triggering	68
Chapter 7. Administering remote MQSeries objects	71
Understanding channels and remote queuing	71
Remote administration	72
Creating a local definition of a remote queue	78
Using remote queue definitions as aliases	81
Chapter 8. Security	83
Before you begin	83
Why you need to protect MQSeries resources	84
Understanding the Object Authority Manager	84
Using the Object Authority Manager commands	87
Object Authority Manager guidelines	89
Understanding the authorization specification tables	93
Understanding authorization files	98
Chapter 9. Using the name service	103
Using DCE to share queues on different queue managers	103
DCE configuration	104
Chapter 10. The MQSeries dead-letter queue handler	107
Invoking the DLQ handler	107
The DLQ handler rules table	108
How the rules table is processed	115
An example DLQ handler rules table	117
Chapter 11. Instrumentation events	119
What instrumentation events are	119
Why use events?	120

Chapter 12. Recovery and restart	125
Making sure that messages are not lost (logging)	125
Checkpointing – ensuring complete recovery	128
Managing logs	130
Using the log for recovery	132
Backup and restore	134
Recovery scenarios	136
Chapter 13. Configuration files	139
What configuration files are	139
MQSeries configuration file	139
Queue manager configuration file	141
Editing configuration files	143
Configuring the logs	143
Specifying log file sizes	147
Chapter 14. Problem determination	149
Preliminary checks	149
Common programming errors	152
What to do next	153
Application design considerations	156
Incorrect output	157
Error logs	160
Dead-letter queues	163
Configuration files and problem determination	164
Using MQSeries trace	164
First failure support technology (FFST)	165
Problem determination with clients	166

Part 2. Reference	169
Chapter 15. MQSeries control commands	171
Names	171
How to read syntax diagrams	171
Syntax help	172
MQSeries return codes	173
crtmqcvx (Data conversion)	174
crtmqm (Create queue manager)	176
dlmqm (Delete queue manager)	180
dspmqaut (Display authority)	182
dspmqcsv (Display command server)	186
dspmqfls (Display MQSeries files)	187
dspmqtrc (Display MQSeries formatted trace output)	189
endmqcsv (End command server)	190
endmqm (End queue manager)	192
endmqtrc (End MQSeries trace)	194
rcdmqimg (Record media image)	195
rcrmqobj (Recreate object)	197
rsvmqtrn (Resolve MQSeries transactions)	199
runmqchi (Run channel initiator)	201
runmqchl (Run channel)	202
runmqdlq (Run dead-letter queue handler)	203
runmqslr (Run listener)	205
runmqsc (Run MQSeries commands)	206
runmqtrm (Start client trigger monitor)	209
runmqtrm (Start trigger monitor)	210
setmqaut (Set/reset authority)	211
strmqcsv (Start command server)	217
strmqm (Start queue manager)	218
strmqtrc (Start MQSeries trace)	219
Part 3. Appendixes	223
Appendix A. MQSeries for Digital OpenVMS at a glance	225
Appendix B. System defaults	227
Appendix C. Directory structure	229
Queue manager log directory structure	231
Appendix D. Sample MQI programs and MQSC files	233
Appendix E. Codeset support on MQSeries for Digital OpenVMS	235
Appendix F. Fast messages and trusted applications.	237
Fast messages	237
Trusted applications	237
Restrictions on Trusted Applications	239
Building applications	240
Integrity	240

Appendix G. Code page conversion tables	241
Code page conversion tables	242
MVS single byte conversion support	268
Appendix H. Stopping and removing queue managers manually	275
Stopping queue managers manually	275
Removing queue managers manually	276
Appendix I. Building your application on Digital OpenVMS	279
MQSeries for Digital OpenVMS data definition files	279
Preparing C programs	283
Preparing COBOL programs	283
User exits	284
Appendix J. Monitoring and controlling DQM on OS/2, Windows NT, UNIX systems and Digital OpenVMS	285
The DQM channel control function	285
Functions available	286
Getting started	288
Channel attributes and channel types	291
Channel functions	292
Appendix K. Setting up communication in Digital OpenVMS systems	297
Deciding on a connection	297
Defining a TCP/IP connection	297
Receiving channels using Attachmate** PathWay for OpenVMS	299
Defining an LU 6.2 connection	300
Appendix L. Defining DECnet Phase IV and Phase V connections	305
Appendix M. Ancilliary information	307
MQSeries Command Reference	307
MQSeries Distributed Queuing Guide	307
Appendix N. Messages	309
Message format	309
Structure of messages	309
MQSeries messages	310
Appendix O. Notices	345
Trademarks	346
<hr/>	
Part 4. Glossary and index	347
Glossary of terms and abbreviations	349
Index	359

Contents

Figures

1.	Example installation	15
2.	Queues, messages, and applications	49
3.	Extract from the MQSC command file, myprog.in	54
4.	Extract from the MQSC report file, myprog.out.	55
5.	Remote administration	73
6.	Setting up channels and queues for remote administration	74
7.	An example rule from a DLQ handler rules table	110
8.	Understanding instrumentation events	120
9.	Monitoring queue managers across different platforms, on a single node	121
10.	Checkpointing	128
11.	Checkpointing with a long-running transaction	129
12.	Example MQSeries configuration file	140
13.	Example queue manager configuration file	142
14.	Sample MQSeries for Digital OpenVMS trace	165
15.	Sample MQSeries for Digital OpenVMS First Failure Symptom Report.	166
16.	Default directory structure after a queue manager has been started	229
17.	Example of COBOL code for including the CMQMDV copy file	281
18.	Example of COBOL code for including two instances of CMQMDV	281
19.	Example of COBOL code for declaring constants	282
20.	Example of COBOL code using the GLOBAL clause	282

Tables

1.	Commands for queue manager administration	36
2.	Commands for command server administration	36
3.	Commands for queue administration	36
4.	Commands for process administration	37
5.	Commands for channel administration	37
6.	Other control commands	37
7.	Security authorization needed for MQI calls	94
8.	MQSC commands and security authorization needed	96
9.	PCF commands and security authorization needed	97
10.	Log overhead sizes	147
11.	How to read syntax diagrams	172
12.	Security authorities from the dspmqaut command	183
13.	Specifying authorizations for different object types	214
14.	Objects included in amqscoma.tst	227
15.	MQSC command files	233
16.	Sample programs - source files	233
17.	Miscellaneous files	234
18.	Locales and CCSIDs	235
19.	Conversion support: US ENGLISH	243
20.	Conversion support: GERMAN	244
21.	Conversion support: DANISH and NORWEGIAN	245
22.	Conversion support: FINNISH and SWEDISH	246
23.	Conversion support: ITALIAN	247
24.	Conversion support: SPANISH	248

Tables

25.	Conversion support: UK ENGLISH / GAELIC	249
26.	Conversion support: FRENCH	250
27.	Conversion support: MULTILINGUAL	251
28.	Conversion support: PORTUGUESE	252
29.	Conversion support: ICELANDIC	253
30.	Conversion support: EASTERN EUROPEAN Languages	254
31.	Conversion support: CYRILLIC	255
32.	Conversion support: GREEK	256
33.	Conversion support: TURKISH	257
34.	Conversion support: HEBREW	258
35.	Conversion support: ARABIC	259
36.	Conversion support: JAPANESE LATIN SBCS	260
37.	Conversion support: JAPANESE KATAKANA SBCS	261
38.	Conversion support: JAPANESE KANJI / LATIN MIXED	262
39.	Conversion support: JAPANESE KANJI / KATAKANA MIXED	263
40.	Conversion support: KOREAN	264
41.	Conversion support: SIMPLIFIED CHINESE	265
42.	Conversion support: TRADITIONAL CHINESE	266
43.	Codeset names and CCSIDs	267
44.	MVS V1.1.4 single byte CCSID conversion support.	268
45.	C include files for MQSeries for Digital OpenVMS	279
46.	COBOL copy files for MQSeries for Digital OpenVMS	281
47.	Functions available in MQSeries for OS/2, Windows NT, UNIX systems and Digital OpenVMS systems	286
48.	Channel attributes for the channel types in OS/2, Windows NT, UNIX systems and Digital OpenVMS	291

About this book

MQSeries for Digital OpenVMS Version 2.2—referred to in this book as MQSeries for Digital OpenVMS or simply MQSeries, as the context permits—is part of the MQSeries family of products. These products provide application programming services that enable application programs to communicate with each other using *message queues*. This form of communication is referred to as *commercial messaging*. The applications involved can exist on different nodes on a wide variety of machine and operating system types. They use a common application programming interface, called the Message Queuing Interface or MQI, so that programs developed on one platform can be readily transferred to another.

This book describes the system administration aspects of MQSeries for Digital OpenVMS Version 2.2 and the services it provides to support commercial messaging in an OpenVMS environment. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Who this book is for

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

What you need to know to understand this book

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

At the back of the book there are some appendixes giving information on the following topics:

- Building applications
- Monitoring, controlling and setting up communications
- Code page conversion tables
- Other ancilliary information you may need

This information has been taken from other books in the MQSeries library and will be reinstated in the appropriate books at the next available opportunity.

How to use this book

Read Chapter 1, “Introduction” on page 3 first for an understanding of MQSeries for Digital OpenVMS.

The sections of this book contain information about:

- How to manage your MQSeries system, including information on:
 - Install and set up the product

MQSeries publications

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

MQSeries publications

Evaluating products

MQSeries Brochure, G511-1908

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

MQSeries Message Queue Interface Technical Reference, SC33-0850

Planning

MQSeries Planning Guide, GC33-1349

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

MQSeries for OS/400 Version 3 Release 2 (and later) Licensed Program Specifications, GC33-1360 (softcopy only)

Administration

MQSeries Clients, GC33-1632

MQSeries Command Reference, SC33-1369

MQSeries Programmable System Management, SC33-1482

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

MQSeries for AT&T GIS UNIX Version 2.2 System Management Guide, SC33-1642

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

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

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

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

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

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

MQSeries for MVS/ESA Version 1 Release 1.4 Program Directory

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

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

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

MQSeries link for R/3 Version 1.0 User's Guide, GC33-1934

MQSeries Three Tier Administration Guide, SC33-1451

MQSeries Three Tier Reference Summary, SX33-6098

MQSeries for Windows Version 2.0 User's Guide, GC33-1822

Application programming

MQSeries Application Programming Guide, SC33-0807

MQSeries Application Programming Reference, SC33-1673

MQSeries Application Programming Reference Summary, SX33-6095

MQSeries for OS/400 Version 3 Release 2 (and later) Application Programming Reference (RPG), SC33-1362

MQSeries Three Tier Application Design, SC33-1636

MQSeries Three Tier Application Programming, SC33-1452

MQSeries Three Tier Reference Summary, SX33-6098

Problem determination

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

MQSeries for AT&T GIS UNIX Version 2.2 System Management Guide, SC33-1642

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

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

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

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

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

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

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

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

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

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

MQSeries Three Tier Administration Guide, SC33-1451

Special topics

MQSeries Distributed Queuing Guide, SC33-1139

Other MQSeries Version 1 publications

For information about other MQSeries platforms, 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 Digital VMS VAX Version 1 Release 5 User's Guide, SC33-1144

MQSeries for SCO UNIX Version 1 Release 4 User's Guide, SC33-1378

MQSeries for Tandem NonStop Kernel Version 1 Release 5.1 User's Guide, SC33-1755

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

MQSeries for VSE/ESA Version 1 Release 4 Licensed Program Specifications, GC33-1483

MQSeries for VSE/ESA Version 1 Release 4 User's Guide, SC33-1142

Information about MQSeries on the Internet

The MQSeries home page

The URL of the MQSeries product family home page is:

<http://www.hursley.ibm.com/mqseries/>

Part 1. Guidance

Chapter 1. Introduction

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

- "MQSeries and message queuing"
- "Messages and queues" on page 4
- "Objects" on page 5
- "System default objects" on page 10
- "Administration" on page 10
- "Clients and servers" on page 11
- "Extending queue manager facilities" on page 11
- "Security" on page 12

MQSeries and message queuing

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

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

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

Time-independent applications

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

Message-driven processing

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

Messages and queues

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

What messages are

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

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

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

Message lengths

In MQSeries for Digital OpenVMS, the maximum message length is 4 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 queues are

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

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

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

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

Predefined and dynamic queues

Queues can be characterized by the way they are created:

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

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

Retrieving messages from queues

In MQSeries for Digital OpenVMS, suitably authorized applications can retrieve messages from a queue according to these retrieval algorithms:

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

The MQGET request from the application determines the method used.

Objects

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

- Queue managers; see “MQSeries queue managers” on page 6.
- Queues; see “MQSeries queues” on page 7.
- Process definitions; see “Process definitions” on page 9.
- Channels; see “Channels” on page 10.

Object names

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

For the other types of objects, each object has a name associated with it and can be referenced in MQSeries for Digital OpenVMS 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.

Objects

In MQSeries, names can have a maximum of 48 characters, with the exception of *channels*, that have a maximum of 20 characters. For more information about names see “Names” on page 171.

Managing objects

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

- MQSeries commands (MQSC), which can be typed in from a keyboard or read from a file.
- Programmable Command Format (PCF) commands, which can be used in a program.

For more information, see Chapter 4, “Understanding administration command sets” on page 33.

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 simply a queue that belongs to another queue manager. A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager may exist on a remote machine across the network or it may exist on the same machine as the local queue manager. MQSeries for Digital OpenVMS supports multiple queue managers on the same machine.

MQI calls

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

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

MQSeries queues

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

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

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

Using queue objects

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

1. A *local queue* object identifies a local queue belonging to the queue manager to which the application is connected. All queues are local queues in the sense that each queue belongs to a queue manager and, for that queue manager, the queue is a local queue.
2. A *remote queue object* identifies a queue belonging to another queue manager. This queue must be defined as a local queue to that queue manager. The information you specify when you define a remote queue object allows the local queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.

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

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

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

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

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

Specific local queues used by MQSeries

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

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

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

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

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

See “Managing objects for triggering” on page 68, and “runmqtrm (Start trigger monitor)” on page 210. For more information about triggering, see the *MQSeries Application Programming Guide*.

Transmission queues: A *transmission queue* temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration; see “Remote administration” on page 72. For information about the use of transmission queues in distributed queuing, see the *MQSeries Distributed Queuing Guide*.

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

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

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

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

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

Event queues: MQSeries for Digital OpenVMS supports instrumentation events, which can be used to monitor queue managers independently of MQI applications. instrumentation events can be generated in several ways, for example:

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

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

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

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

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries for Digital OpenVMS queue manager. See “Initiation queues” on page 8 for more information.

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

Use the MQSC command DEFINE PROCESS or the PCF command Create Process to create a process definition.

Channels

Channels are objects that provide a communication path from one queue manager to another. Channels are used in distributed message queuing to move messages from one queue manager to another. They shield applications from the underlying communications protocols. The queue managers may exist on the same, or different, platforms. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another, complementary one, at the queue manager that is to receive them.

For information on channels and how to use them, see the *MQSeries Distributed Queuing Guide*, and also “Preparing channels and transmission queues for remote administration” on page 73.

System default objects

The *system default objects* are a set of object definitions that can be created for each queue manager, using the command file `amqscoma.tst`, which is supplied with MQSeries. You can copy and modify any of these object definitions for use in applications at your installation. Default object names have the stem `SYSTEM.DEF`; for example, the default local queue is `SYSTEM.DEFAULT.LOCAL.QUEUE`; the default receiver channel is `SYSTEM.DEF.RECEIVER`. You cannot rename these objects; default objects of these names are required.

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

Administration

In MQSeries, you carry out administration tasks by issuing *commands*. Three command sets are provided, depending on which tasks you want to perform and how you want to perform them. The command sets are described in Chapter 4, “Understanding administration command sets” on page 33. Administration tasks include:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This is described in detail in the *MQSeries Distributed Queuing Guide*.

Local and remote administration

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

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

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

Clients and servers

MQSeries for Digital OpenVMS 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 the *MQSeries Distributed Queuing Guide*

MQSeries applications in a client-server environment

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

Extending queue manager facilities

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

- User exits
- Installable services

User exits

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

- *Channel exits*, which change the way that channels operate.
- *Data conversion exits*, that can be called from application programs to convert data from one format to another.

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

Installable services

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

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

- The **authorization service**, which allows you to build your own security facility. The default service component that implements the service is the Object Authority Manager (OAM), which is supplied with the product. By default, the OAM is active, that is, you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM.
- The **name service**, which allows queue managers to share queues. The default service component that implements the service uses the Open Software Foundation (OSF) Distributed Computing Environment (DCE). This component enables a queue manager to determine the owner of a queue.

You can also write your own name service component, for example, if you do not have DCE.

See Chapter 9, “Using the name service” on page 103 and also the *MQSeries Programmable System Management* manual.

Security

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

Chapter 2. Installing MQSeries for Digital OpenVMS

This chapter tells you how to install MQSeries for Digital OpenVMS and how to verify that your installation has been successful. It contains the following topics:

- “Components you can install”
- “Installation requirements” on page 14
- “Installation on Digital OpenVMS Version 6.2” on page 15
- “Post-installation tasks” on page 19
- “System configuration” on page 21
- “Directories that exist after installation” on page 21
- “Translated messages” on page 22
- “Verifying your installation” on page 22
- “Installing clients” on page 26

The MQSeries product is installed into SYS\$SYSTEM and various other places including SYS\$SYSROOT, SYS\$HELP, SYS\$STARTUP, and SYS\$LIBRARY.

MQS_ROOT:[MQM] contains the data files. MQS_ROOT is a logical name whose location is decided at install time.

Components you can install

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

Base product and runtime

The MQSeries for Digital OpenVMS base code.

Server

Support for servers. Requires the base product to be installed.

Help

Help files for control commands, MQI calls, and MQSC commands.

OpenVMS client

Support for Digital OpenVMS clients.

Desktop clients

Support for:

- DOS clients with a Digital OpenVMS server. To install a DOS client, the DOS code must be copied to the DOS client machine or machines.
- OS/2 clients with a Digital OpenVMS server. To install an OS/2 client, the OS/2 code must be copied to the OS/2 client machine or machines.
- Windows 3.1 clients with a Digital OpenVMS server. To install a Windows client, the Windows code must be copied to the Windows 3.1 client machine or machines.

Samples

Samples

French

MQSeries messages - French

Installation requirements

German

MQSeries messages - German

Japanese

MQSeries messages - Japanese

Spanish

MQSeries messages - Spanish

Typically, a particular Digital OpenVMS machine is designated as either an MQSeries client or server. This means that in most cases you install either the Digital OpenVMS client component or the base product and server component.

Installation requirements

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

Disk storage

These are the approximate storage requirements:

- Base code and server product code and data:
 - VAX – 16 MB of disk space
 - AXP – 18 MB of disk space
- Clients:

If you are installing client code, the storage required on the client machines is:

DEC-OVMS

820 KB on the VAX machine

935 KB on the AXP machine

DOS 240 KB on the DOS machine

OS/2 1.2 MB on the OS/2 machine

Windows 552 KB on the Windows machine

Working data for MQSeries for Digital OpenVMS is stored by default in `MQS_ROOT:[MQM]`. See Figure 16 on page 229 for the directory structure.

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

Memory requirements

- VAX – 16 MB of memory
- AXP – 32 MB of memory

Disk quotas

If disk quotas are enabled on the volume where MQSeries is going to store its data (that is, the MQS_ROOT device) the MQM identifier requires a quota.

Note: First failure support technology (FFST) files also use disk quota.

Installation on Digital OpenVMS Version 6.2

Use the Digital OpenVMS **VMSINSTAL** program. For further details see the *Digital OpenVMS System Manager's Guide*.

From the OpenVMS command prompt type:

```
$ @SYS$UPDATE:VMSINSTAL MQSERIES022 device
```

where device is the location for the installation save sets, for example dka400:[MQSERIES022.KIT].

Select the items you require when prompted. Refer to Figure 1 for examples.

Note: The words in the square parentheses are the default selections. To accept the default, press the Enter key. To change the selection, type **Y** for yes, or **N** for no, as appropriate.

```
$ @sys$update:vmsinstal

      OpenVMS AXP Software Product Installation Procedure V6.2

It is 14-MAR-1997 at 12:14.

Enter a question mark (?) at any time for help.

* Are you satisfied with the backup of your system disk [YES]?
* Where will the distribution volumes be mounted:DKA400:[MQSERIES022.KIT]

Enter the products to be processed from the first distribution volume set.
* Products: mqseries022
* Enter installation options you wish to use (none):

The following products will be processed:

      MQSERIES V2.2

      Beginning installation of MQSERIES V2.2 at 12:14

%VMSINSTAL-I-RESTORE, Restoring product save set A ...
%VMSINSTAL-I-REMOVED, Product's release notes have been moved to SYS$HELP.
* Do you want to purge files replaced by this installation [YES]?

Licensed Materials - Property of IBM

5697-270/5697-271
  (C) Copyright International Business Machines Corp. 1994, 1996

All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.
```

Figure 1 (Part 1 of 5). Example installation

Installation procedure

```
Please choose which of the following components to install:

- all the MQSeries components
- MQSeries Client for OpenVMS (684 blocks)
- MQSeries Server (10852 blocks)
- MQSeries Base Kit for Client and Server (1180 blocks)
- MQSeries Runtime for Client and Server (11204 blocks)
- MQSeries Examples (2044 blocks)
- MQSeries Message Catalogs - Spanish (424 blocks)
- MQSeries Message Catalogs - French (424 blocks)
- MQSeries Message Catalogs - German (444 blocks)
- MQSeries Message Catalogs - Japanese (364 blocks)
- MQSeries Clients for OS/2, DOS, and Windows 3.1 (9308 blocks)

* Do you want all the MQSeries components [NO]? YES
* Are you sure you want to install all the options [YES]?
%MQSERIES-I-FREEBLKS, Checking for 26440 free blocks
* Do you want to run the IVP after the installation [YES]?
%MQSERIES-I-UCX, TCP/IP Services for OpenVMS is installed
%MQSERIES-I-LU62, SNA LU6.2 Services for OpenVMS is installed
* Enter the root device for the MQSeries datafiles
[SYS$COMMON]: DKA0
%VMSINSTAL-I-ACCOUNT, This installation adds an identifier named MQM.
%UAF-I-RDBADDMSG, identifier MQM value %X8001001D added to rights database
%UAF-I-GRANTMSG, identifier MQM granted to SYSTEM

*****
The installation procedure will create an account called MQM
to run the MQSeries server processes. The account will be
created with the MQM resource identifier granted and the
following privileges:
TMPMBX,NETMBX

You must specify a unique group UIC for this account in
order to ensure proper security of the network. The
password for this account will be generated. You do not
need to know the password, since the account is disabled.
If this scenario violates your security policies, you may
change it after the installation has finished via the
OpenVMS AUTHORIZE utility.
*****

* Enter the UIC of the new MQM account [400,400]:
%VMSINSTAL-I-ACCOUNT, This installation creates an ACCOUNT named MQM.
%UAF-I-ADDMSG, user record successfully added
%VMSINSTAL-I-ACCOUNT, This installation adds an identifier named
MQS_SERVER.
%UAF-I-RDBADDMSGU, identifier MQS_SERVER value [000400,000400]
added to rights database
%VMSINSTAL-I-SYSDIR, This product creates system specific directory
[MQS_SERVER].
%VMSINSTAL-I-ACCOUNT, This installation updates an ACCOUNT named MQM.
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-GRANTMSG, identifier MQM granted to MQS_SERVER
```

Figure 1 (Part 2 of 5). Example installation

All questions have been answered. MQSeries installation for OpenVMS will continue.

```
%VMSINSTAL-I-RESTORE, Restoring product save set B ...
%VMSINSTAL-I-RESTORE, Restoring product save set C ...
%VMSINSTAL-I-SYSDIR, This product creates system directory
[MQS_INCLUDE].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.BIN].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DLQ].
%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...
%VMSINSTAL-I-RESTORE, Restoring product save set E ...
%VMSINSTAL-I-RESTORE, Restoring product save set F ...
%VMSINSTAL-I-RESTORE, Restoring product save set G ...
%VMSINSTAL-I-RESTORE, Restoring product save set H ...
%VMSINSTAL-I-RESTORE, Restoring product save set I ...
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.MSG].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.LIB].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.BIN].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.INC].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.SAMP.BIN].
%VMSINSTAL-I-RESTORE, Restoring product save set J ...
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.MSG].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.DLL].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.BIN].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.LIB].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.INC].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.SAMP.BIN].
%VMSINSTAL-I-RESTORE, Restoring product save set K ...
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.MSG].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.DLL].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.LIB].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.BIN].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.INC].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.SAMP].
%VMSINSTAL-I-SYSDIR, This product creates system directory
[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.SAMP.BIN].
```

Figure 1 (Part 3 of 5). Example installation

Installation procedure

```
*****
The MQSeries environment is setup by the command procedure
SYS$STARTUP:MQS_STARTUP.COM. It should be invoked during
system startup to define the MQSeries system logicals and
install all MQSeries shared libraries.

The following command line can be added to the system
startup command file SYS$MANAGER:SYSTARTUP_VMS.COM:

$ @SYS$STARTUP:MQS_STARTUP.COM

Or the MQSeries command file can be executed from SYSMAN by
adding it to the startup command list with the following
command:

$ MCR SYSMAN STARTUP ADD FILE MQS_STARTUP.COM
*****

The installation will now execute the installation verification procedure
***Creating the IVP queue manager
MQSeries queue manager created.
***Starting the IVP queue manager
MQSeries queue manager started.
***Creating the base queues with RUNMQSC
83H8439, 5697-270 (C) Copyright IBM Corp. 1996. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8010: MQSeries process created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8014: MQSeries channel created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
AMQ8006: MQSeries queue created.
21 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
***Creating the IVP Test queue
83H8439, 5697-270 (C) Copyright IBM Corp. 1996. ALL RIGHTS RESERVED.
Starting MQSeries Commands.
```

Figure 1 (Part 4 of 5). Example installation

```

AMQ8006: MQSeries queue created.
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
***Writing to the IVP Test queue
Sample AMQSPUT0 start
target queue is testq
Sample AMQSPUT0 end
***Reading from the IVP Test queue
Sample AMQSGET0 start
message <This is an IVP test message being read from the test queue.>
no more messages
Sample AMQSGET0 end
***Ending the IVP queue manager
MQSeries queue manager ending.
MQSeries queue manager ended.
***Deleting the IVP queue manager
MQSeries queue manager deleted.
***IVP Completed Successfully

      Installation of MQSERIES V2.2 completed at 12:24

      Adding history entry in VMI$ROOT:[SYSUPD]VMSINSTAL.HISTORY

      Creating installation data file: VMI$ROOT:[SYSUPD]MQSERIES022.VMI_DATA

Enter the products to be processed from the next distribution volume set.
* Products:

      VMSINSTAL procedure done at 12:24

$

```

Figure 1 (Part 5 of 5). Example installation

Post-installation tasks

After installing MQSeries you may want to do one or more of the following:

- Set up one or more separate MQSeries Administrator accounts
- Create identifiers for groups that use MQSeries

Setting up a separate MQSeries Administrator account

MQSeries Administration can be performed through the SYSTEM account on Digital OpenVMS. The MQSeries installation procedure provides all required quotas and grants all required privileges to the SYSTEM account for this purpose.

However, rather than the VMS System Manager, you may want to have another person, or just a separate account, to administer MQSeries functions at your enterprise.

You must perform the following steps to set up the MQSeries Administrator account:

1. Use the Digital OpenVMS Authorize utility to setup an interactive account as your MQSeries Administrator, with the identical privileges and quotas as the MQM account created by the installation procedure.

Note: No special privileges are required.

In this example the name of the account is MQADMIN.

2. Grant the MQM identifier to your MQSeries Administrator as follows:
 - a. \$ RUN AUTHORIZE
 - b. UAF> GRANT/IDENTIFIER/ATTRIBUTE=RESOURCE MQM MQADMIN
 - c. Exit authorize using <Ctrl Z>

In the preceding example, the name of the identifier being granted to the account is MQM and the actual user name being granted to the administrator account is MQADMIN, set in Step 1.

Note: You can verify that you have set up the account correctly, using the command:

```
$ @SYS$MANAGER:MQS_CHECKADMIN
```

Creating identifiers for groups that use MQSeries

The MQM identifier is created during installation and essentially grants access to MQSeries Administrative functions. If MQSeries security is being used, you will need to create additional identifiers to represent the groups of VMS accounts that can be granted access to MQSeries objects. These identifiers will be granted to application groups using the VMS Authorize utility.

See Chapter 8, “Security” on page 83 for further information about using MQSeries security features.

For example, users whose VMS accounts are in different UIC groups may want to share MQSeries resources, for example, queues. The users of these common queues may be granted the identifier called PAYROLL. To do this, you:

1. Add the PAYROLL identifier as a resource as follows:
 - a. \$ RUN AUTHORIZE
 - b. UAF> ADD/IDENTIFIER/ATTRIBUTE=RESOURCE PAYROLL
 - c. Exit authorize using <Ctrl Z>.
2. Grant the PAYROLL identifier to the desired user accounts (in this case, DOMESTIC and OVERSEAS) as follows:
 - a. \$ RUN AUTHORIZE
 - b. UAF> GRANT/IDENTIFIER/ PAYROLL DOMESTIC
 - c. UAF> GRANT/IDENTIFIER/ PAYROLL OVERSEAS
 - d. Exit authorize using <Ctrl Z>.
3. Grant appropriate MQSeries authorizations for the grouped user accounts, using the SETMQAUT command, according to the capabilities required:

```
setmqaut -m qm0 -t qmgr -g payroll +connect  
setmqaut -m qm0 -t queue -n 401k.q -g payroll +inq +put +get
```

Use +connect to allow the user group to connect to a desired queue manager.
+inq, +put, +get to allow the user group to inquire upon, put messages to, and get messages from a desired queue.

System configuration

MQSeries makes use of shared memory to communicate between processes and it is, therefore, likely that the default SYSGEN configuration is not adequate.

The minimum recommended values of the SYSGEN parameters are:

GBLSECTIONS

Add 15 for each queue manager and 2 for each defined queue

GBLPAGES

Add 4000 for each queue manager

GBLPAGFIL

Set to at least 8000 on Alpha and 16000 on VAX.

Add 4000 for each queue manager after the first two.

See the *OpenVMS System Manager's Manual* for details on how to change these values.

System limitations

MQSeries for Digital OpenVMS V2.2 has a limitation on the number of processes that MQSeries can support on a single machine. Each queue manager requires three or four basic processes, and each application or channel that connects to a queue manager normally requires two processes – one for the application (or channel) and one for its agent process.

See Appendix F, “Fast messages and trusted applications.” on page 237 for details of trusted applications, which eliminate the need for agent processes.

The limitations on processes are as follows:

Number of queue managers	Number of applications or channels
1	124
2	120
5	108
10	84

Directories that exist after installation

When you install MQSeries for Digital OpenVMS, the following directories are created (if required) and populated with the MQSeries product files:

SYS\$SYSTEM	All executable files
SYS\$LIBRARY	All shared library files
MQS_ROOT: [MQM]	Base directory for all queue manager and queue files
MQS_INCLUDE	Include files for building MQSeries applications
MQS_EXAMPLES	Source files for sample programs

Under MQS_EXAMPLES are the following directories:

[.BIN]	Executable versions of the sample programs
[.DLQ]	Sample dead letter queue handling source
[.OS2_CLIENT]	Root for the OS/2 MQSeries client
[.DOS_CLIENT]	Root for the DOS MQSeries client
[.WIN_CLIENT]	Root for the Windows 3.1 MQSeries client

Verifying your installation

When you install the product you also receive additions to the HELPLIB.HLP file to supply help for the following topics:

- MQSeries control commands
- MQSeries commands (MQSC)
- Message Queue Interface (MQI) calls

Also provided on the product CD under the [MQSERIES022.DOCUMENTATION] directory is a set of MQSeries books in PostScript format, enabling you to print the books on a Postscript printer, or view the book with a suitable viewer. The filenames of the files and the names of the corresponding books are:

Filename	Book
MQAPG.PS	<i>MQSeries Application Programming Guide</i>
MQAPRM.PS	<i>MQSeries Application Programming Reference</i>
MQAPRS.PS	<i>MQSeries Application Programming Reference Summary</i>
MQCLIENT.PS	<i>MQSeries Clients</i>
MQCMDREF.PS	<i>MQSeries Command Reference</i>
MQDQMGDE.PS	<i>MQSeries Distributed Queuing Guide</i>
MQPLNGDE.PS	<i>MQSeries Planning Guide</i>
MQPSM.PS	<i>MQSeries Programmable System Management</i>
MQOVMSMG.PS	<i>MQSeries for Digital OpenVMS Version 2 System Management Guide</i>

Translated messages

Messages in US English are always available. If you require another of the languages that is supported by MQSeries for Digital OpenVMS Version 2, you **must** ensure that your SYS\$NLSPATH logical name includes the appropriate directory. This is normally done automatically by the MQSeries startup procedure. Furthermore, the SYS\$LC_ALL logical name must specify the correct locale for the language, country and codeset.

For example, to select messages in German:

```
$ DEFINE/SYSTEM SYS$LC_ALL DE_DE_IS08859-1.LOCALE
```

Verifying your installation

After installation the "MQM" identifier owns the directories and files that contain the resources associated with the product. The "MQM" resource and "MQM" user account must be defined on any machine on which the MQSeries software is to be installed, whether the machine is a client or a server machine.

If you want to run any administration commands, for example, crtmqm (create queue manager) or strmqm (start queue manager), your account must be granted the "MQM" identifier by the system manager. The installation procedure creates the "MQM" identifier and grants it to the system manager as well as to the "MQM" account.

When you have installed the MQSeries for Digital OpenVMS base, server and samples components, you should verify that the installation has completed successfully.

Note: The installation process automatically runs the Installation Verification Procedure (see “Installation on Digital OpenVMS Version 6.2” on page 15) which reports the success or failure of basic MQSeries operation.

You should also perform these steps manually, using the MQSC command file amqscoma.tst, as described on page 24. The commands in this file initialize your MQSeries system and set up the default objects that your system requires. The objects that amqscoma.tst creates for you are listed in Appendix B, “System defaults” on page 227.

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

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

Case sensitivity

MQSeries object names are case sensitive. For example, MYQUEUE and MyQueue are unique names for different queues.

In contrast, Digital OpenVMS does not automatically preserve the case of parameters passed to commands. All parameters are by default converted by the DCL command interpreter to uppercase. For example:

```
$ crtmqm A.QUEUE.MGR
```

and

```
$ crtmqm a.queue.mgr
```

perform the same operation, that is, create a queue manager with the name “A.QUEUE.MGR” in uppercase.

To enter a lowercase or mixedcase name as a parameter, it must be enclosed in quotation marks. For example:

```
$ crtmqm “New.Queue.Mgr”
```

creates a queue manager with the name in mixed case.

Invoking MQSeries commands from DCL

MQSeries commands are implemented as DCL “foreign” commands. You should note that the names of DCL commands are not case sensitive.

In order to invoke MQSeries commands, which reside in the SYS\$SYSTEM directory) as if they were native DCL commands, you **must** invoke the command file SYS\$MANAGER:MQS_SYMBOLS.COM in the system-wide login file SYS\$MANAGER:SYSLOGIN.COM, or in the login files for all users who need to issue MQSeries commands.

Redirection of **SYS\$INPUT**, **SYS\$OUTPUT**, and **SYS\$ERROR**

To improve the ease of migration from other operating systems to VMS, MQSeries supports the UNIX style of redirection indicators for `sys$input`, `sys$output`, and `sys$error`, as follows:

- < specifies the source for `SYS$INPUT`
- > specifies the output for `SYS$OUTPUT`
- > > specifies the output for `SYS$ERROR`

Follow these steps to verify your installation

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

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

```
crtmqm -q QMNAME
```

Notes:

- a. The queue manager name is case-sensitive.
 - b. The `-q` flag denotes that this is the default queue manager. It is not essential to have a default queue manager, but it is recommended practice to have one.
 - c. For a detailed description of the `crtmqm` command and options see “`crtmqm` (Create queue manager)” on page 176.
2. Start the queue manager using this command:

```
strmqm QMNAME
```

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

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

```
runmqsc QMNAME < MQS_EXAMPLES:AMQSCOMA.TST > DEF OBJ. OUT
```

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

```
AMQ8006 MQSeries queue created
```

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

You may want to modify a copy of AMQSCOMA.TST to meet your own requirements for system defaults.

4. Stop the queue manager using the command:

```
endmqm QMNAME
```

5. Delete the queue manager using the command:

```
dltmqm QMNAME
```

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

If your installation was not a success?

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

- **Did you type in the commands correctly?**

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

- **Do you have enough disk space or memory to run the verification?**

Check any error messages for an indication of this. If error message AMQ7065 Insufficient space on disk is returned, use the SHOW DEVICE command to display the free blocks on the MQS_ROOT device. If there are fewer than 32,000 free blocks, you must free some space on that device.

- **Do the required directories for the installed product exist?**

If they do not exist, attempt to reinstall.

- **Do you have the required authority to run the commands?**

Check that your account has been granted the MQM resource by entering the SHOW PROCESS/PRIVILEGE command.

- **Have you performed a SYSGEN to prepare your system for MQSeries**

See "System configuration" on page 21 for more information.

Installing clients

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

- OS/2
- DOS
- Windows 3.1

You can install Digital OpenVMS client software directly from CD-ROM or from the LAN.

Note: For VAX machines this software is also available on tape.

To install client software, other than that for Digital OpenVMS install the component on a Digital OpenVMS machine in the usual way and then copy the client files to the client platform.

The required files are located in these directories:

- The DOS files are in these directories:

```
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.LIB]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.BIN]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.INC]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.DOS_CLIENT.MSG]
```

- The OS/2 files are in these directories:

```
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.LIB]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.BIN]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.DLL]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.INC]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.OS2_CLIENT.MSG]
```

- The Windows files are in these directories:

```
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.LIB]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.BIN]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.DLL]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.INC]
SYS$COMMON:[SYSHLP.EXAMPLES.MQSERIES.WIN_CLIENT.MSG]
```

Note: The logical name MQS_EXAMPLES is assigned to
SYS\$COMMON:[SYSHLP.EXAMPLES.MQSERIES].

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

Chapter 3. Customizing your system

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

Do I need to customize?

When you have installed the product, you can use it without having to customize it in any way. The default configuration provides all the facilities you need to build a working system that can participate in message queuing with other MQSeries systems.

When do I customize?

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

What are configuration files?

There are two types of configuration files. One contains information about the way your MQSeries system is set up or configured; this file is created when MQSeries is installed. The other contains information about the attributes of an individual queue manager. This file is generated when a queue manager is created.

“Things you can customize” on page 28 specifies which of these files to modify for each relevant configuration task. For more information about the files themselves, see Chapter 13, “Configuration files” on page 139.

What do I do now?

Check each item in the list on page 28 to see if any of the things that you can customize apply to you. If not, you do not need to do anything else at this time and you can go on to the next chapter.

Things you can customize

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

- Implementing data conversion; see page 28.
- Defining the default and system objects; see page 29.
- Changing the location of queue manager objects; see page 30.
- Specifying logging parameters; see page 30.
- Configuring a queue manager; see page 31.
- Specifying user groups for security administration; see page 31.

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

Implementing data conversion

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

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

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

Supported code sets

MQSeries for Digital OpenVMS supports most of the code sets used by the locales that are provided as standard on OpenVMS.

Details of the supported code sets are given in Appendix E, “Codeset support on MQSeries for Digital OpenVMS” on page 235.

Adding information about coded character sets on Digital OpenVMS

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

To update coded character set information edit the file `ccsid.tbl` in the directory `MQS_ROOT:[MQM.CONV.TABLE]`.

Example of ccsid.tbl

```

# Licensed Materials - Property of IBM
#
# (C) Copyright IBM Corp. 1994, 1995
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# This file is used to add newly supported CCSID values to the system.
# To be used the file should be placed in the [MQM.CONV.TABLE]
# directory. CCSID and encoding values can be found in the CDRA
# documentation (SC09-1390). For mixed CCSIDs the "SBCS part" field
# should be set to the SBCS CCSID, otherwise this field should be 0.
# The codeset name is the name by which this CCSID is known on this
# operating system.
#
# The file can also be used to modify data stored for existing CCSIDs.
#
# Decimal      Hexadecimal      Decimal      Codeset
# CCSID        Encoding system  SBCS part    name
850            0x2100           0            ib850
437            0x2100           0            ib437

```

Defining the default and system objects

- *This task is required, but is part of the standard administration procedures, see Chapter 5, "Managing queue managers" on page 39.*

MQSeries for OpenVMS provides an MQSC command file that you can use to set up the default and system objects. Typically, when you define an object, you do not define all the possible attributes. The ones you do not specify are inherited from the corresponding default object. The supplied command file `amqscoma.tst`, when used with the `runmqsc` command, creates a set of default and system objects. See "Running the supplied MQSC command files" on page 55 for information about running this sample.

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

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

Modifying the `amqscoma.tst` command file

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

- You have a large number of objects to create that have similar, but not identical, values to those in the `amqscoma.tst` file.
- You have some specific requirements or limitations on the size of certain resources.

Customizable entities

To modify `amqscoma.tst`, make a backup copy, make the required changes, and then use the new version of the file to create the default objects. See also “Creating the default and system objects” on page 44.

Changing the location of queue manager objects

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

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

If you change this prefix, all the objects are created at the locations specified by the new prefix. Unless you change it, the default prefix for queue manager objects is: `MQS_ROOT:[MQM]`.

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

Changing the default prefix

You can change the default prefix, so that when you create a new queue manager its prefix is taken from the new default. The default prefix is specified in the *DefaultPrefix* stanza in the `mqs.ini` file. Unless you have changed it, the default prefix is: `MQS_ROOT:[MQM]`.

Specifying logging parameters

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

The logging parameters determine the type and size of the logs your system will use. These are specified in the configuration files `mqs.ini` and `qm.ini`, which are read when a queue manager is started. See “Log configuration stanzas” on page 144 for more information.

Note: MQM **must** own the log files and they should contain an ACL allowing full access by the MQM identifier. If you change the locations of these files, you must set the security yourself. This is not required if the log files are in the default locations supplied with the product.

Configuring a queue manager

- *This task is required, but is part of the standard administration procedures. See Chapter 5, “Managing queue managers” on page 39.*

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

Once you have created a queue manager, you may need to modify its properties. For more information, see “Guidelines for creating queue managers” on page 39 and Chapter 13, “Configuration files” on page 139.

Specifying user groups for security administration

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

The user account and resource MQM are created when you install the product.

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

Customizable entities

Chapter 4. Understanding administration command sets

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

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

MQSeries for Digital OpenVMS provides three command sets for invoking administration tasks:

- Control commands
- MQSC commands
- PCF commands

This chapter describes the command sets that are available and provides a summary of the different commands in “Comparing command sets” on page 36.

Control commands

Control commands fall into three categories:

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

Using control commands

You type in control commands at a DCL prompt. The command name is not case sensitive, but the arguments are. For example, in the command:

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

- The flag must be -u, not -U.
- The dead-letter queue is SYSTEM.DEAD.LETTER.QUEUE.
- The argument is specified as “jupiter.queue.manager”; this is different from “JUPITER.queue.manager”.

MQSeries commands

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

Chapter 15, “MQSeries control commands” on page 171 describes the syntax and purpose of each command.

MQSeries commands (MQSC)

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

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

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

These commands are summarized in “Comparing command sets” on page 36. For detailed information about each MQSC command, see *MQSeries Command Reference*.

Running MQSC commands

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

- Interactively by typing them at the keyboard. See “Using the MQSC facility interactively” on page 51.
- As a sequence of commands from an ASCII text file. See “Running MQSC commands from text files” on page 53.

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.

For more information about using the MQSC facility and text files, see “Using the MQSC facility interactively” on page 51. For more information about the **runmqsc** command, see “runmqsc (Run MQSeries commands)” on page 206.

PCF commands

The purpose of the MQSeries programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way you can create queues and process definitions, and change queue managers, from a program. In fact, PCF commands cover the same range of functions that are provided by the MQSC facility. You can therefore write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

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

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

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

Message descriptor

This is a standard MQSeries message descriptor, in which:

- Message type specifies a *management request*.
- Message format specifies *administration commands*.

Application data

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

- The PCF message type specifies *command*.
- The command identifier specifies the command, for example, *Change Queue*.

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

Attributes in MQSC and PCFs

Object attributes specified in MQSC are shown in this book in uppercase, for example RQMNAME, although they are not case-sensitive. These attribute names are limited to eight characters, so it is not easy to work out the meaning of some of them, for example, QDPHIEV. Object attributes in PCF are shown in italics, are not limited to eight characters, and are therefore easier to read. The PCF equivalent of RQMNAME, is *RemoteQMgrName* and of QDPHIEV is *QDepthHighEvent*.

Escape PCFs

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

Comparing command sets

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

Note: Only MQSC commands that apply to MQSeries for Digital OpenVMS are shown.

PCF	MQSC	Control
Change Queue Manager	ALTER QMGR	–
(Create queue manager)*	–	crtmqm
(Delete queue manager)*	–	dltmqm
Inquire Queue Manager	DISPLAY QMGR	–
(Stop queue manager)*	–	endmqm
Ping Queue Manager	PING QMGR	–
(Start queue manager)*	–	strmqm

Note: * Not available as PCF commands.

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

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

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

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

<i>Table 4. Commands for process administration</i>	
PCF	MQSC
Change Process	ALTER PROCESS
Copy Process	DEFINE PROCESS(x) LIKE(y)
Create Process	DEFINE PROCESS
Delete Process	DELETE PROCESS
Inquire Process	DISPLAY PROCESS
Inquire Process Names	DISPLAY PROCESS
Note: There are no equivalent control commands in this group.	

<i>Table 5. Commands for channel administration</i>		
PCF	MQSC	Control
Change Channel	ALTER CHANNEL	–
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	–
Create Channel	DEFINE CHANNEL	–
Delete Channel	DELETE CHANNEL	–
Inquire Channel	DISPLAY CHANNEL	–
Inquire Channel Names	DISPLAY CHANNEL	–
Ping Channel	PING CHANNEL	–
Reset Channel	RESET CHANNEL	–
Resolve Channel	RESOLVE CHANNEL	–
Start Channel	START CHANNEL	runmqchl
Start Channel Initiator	START CHINIT	runmqchi
Start Channel Listener	–	runmqlsr
Stop Channel	STOP CHANNEL	–

<i>Table 6. Other control commands</i>	
Description	Control
Create MQSeries conversion exit	crtmqcvx
Display authority	dspmqaout
Display files used by objects	dspmqlfs
Display MQSeries formatted trace output	dspmqtrc
End MQSeries trace	endmqtrc
Record media image	rcdmqimg
Recreate media object	rcrmjobj
Resolve MQSeries transactions	rsvmqtrn
Run MQSC commands	runmqsc
Run trigger monitor	runmqtrm
Run client trigger monitor	runmqtrmc
Set or reset authority	setmqaut
Start MQSeries trace	strmqtrc
Note: Functions in this group are available only as control commands. There are no direct PCF or MQSC equivalents.	

Comparing command sets

Chapter 5. Managing queue managers

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

- “Getting started”
- “Guidelines for creating queue managers”
- “Understanding MQSeries file names” on page 42
- “Working with queue managers” on page 43
- “Managing the command server for remote administration” on page 47

Getting started

Before you can do anything with messages and queues, you must create at least one queue manager. Once the installation process is complete, you can use the MQSeries control commands to create a queue manager and start it. Then you can use MQSC commands to create the required default objects and system objects. Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. You must create these objects for each queue manager you create. The supplied command file `amqscoma.tst`, when used with the **runmqsc** command, creates a set of default and system objects. See “Running the supplied MQSC command files” on page 55 for information about running this sample.

See Chapter 4, “Understanding administration command sets” on page 33 for more information about commands that can be used with MQSeries for Digital OpenVMS, and the different methods of invoking them.

Guidelines for creating queue managers

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

- Specifying a unique queue manager name.
- Limiting the number of queue managers.
- Specifying a default queue manager.
- Specifying a dead-letter queue.
- Specifying a default transmission queue.
- Specifying the required logging parameters.
- Backing up configuration files after creating a queue manager.

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

Specifying a unique queue manager name

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

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

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

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

Limiting the number of queue managers

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

See “System limitations” on page 21 for further information.

Specifying the default queue manager

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

To create a queue manager use the **crtmqm** command. For a detailed description of this command and its parameters, see “crtmqm (Create queue manager)” on page 176.

What is a default queue manager?

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

How do you specify a default queue manager?

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

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

What happens if I make another queue manager the default?

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

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

Specifying 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.

Attention: 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 53 for an example of an MQSC ALTER command.

A sample dead-letter queue definition is provided with the supplied sample `amqscoma.tst`. The queue is called `SYSTEM.DEAD.LETTER.QUEUE`. See “Creating the default and system objects” on page 44 for information about running this sample. When you find messages on a dead-letter queue, you can use the dead-letter queue handler, supplied with MQSeries, to process these messages. See Chapter 10, “The MQSeries dead-letter queue handler” on page 107 for further information about the dead-letter queue handler itself, and how to reduce the number of messages that might otherwise be placed on the dead-letter queue.

Specifying a default transmission queue

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

When you create a queue manager you should use the `-d` flag to specify the name of the default transmission queue. This does not actually create the queue; you have to do this explicitly later on. See “Working with local queues” on page 58 for more information.

Specifying the required logging parameters

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

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

For more information about specifying logging parameters:

- On the **crtmqm** command, see “crtmqm (Create queue manager)” on page 176.
- Using configuration files, see “Log configuration stanzas” on page 144.

Backing up configuration files after creating a queue manager

There are two configuration files to consider:

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

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

For more information about configuration files, see Chapter 13, “Configuration files” on page 139.

Understanding MQSeries file names

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

- A prefix - the first part of the name:
MQS_ROOT:[MQM]
This prefix is defined in the queue manager configuration file.
- A literal:
QMGRS
- A coded queue manager name, which is the queue manager name transformed into a valid directory name. For example, the queue manager:

QUEUE.MANAGER

would be represented as:

QUEUE\$MANAGER

This process is referred to as *name transformation*.

Queue manager name transformation

In MQSeries you can give a queue manager a name containing up to 48 characters. For example, you could name a queue manager:

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

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

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

1. Transform individual characters:
 - . becomes \$
 - / becomes _
 - % 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 becomes:

```
MQS_ROOT:[MQM.QMGRS.QUEUE$MANAGER]
```

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

Object name transformation

Object names are not necessarily valid file system names. Therefore the object names may need to be transformed. The method used is different from that for queue manager names because, although there only a few queue manager names per machine, there can be a large number of other objects for each queue manager. Only process definitions and queues are represented in the file system; channels are not affected by these considerations.

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

The queue files names start with the letter "Q".

Working with queue managers

MQSeries provides control commands for creating, starting, ending, and deleting queue managers. You can also display a queue manager's attributes using the MQSC command `DISPLAY QMGR` and change them using `ALTER QMGR`. See "Displaying queue manager attributes" on page 51 and "Altering queue manager attributes" on page 53.

Creating a default queue manager

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

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

where:

- q 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. For `crtmqm`, this must be the last parameter in the command.

Starting a queue manager

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

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

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

Creating the default and system objects

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

```
runmqsc "saturn.queue.manager" < MQS_EXAMPLES:amqscoma.tst > defobj.out
```

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

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

For more information about running the MQSC facility (`runmqsc`), see "Running MQSC commands from text files" on page 53.

Looking at object files

Each MQSeries queue, queue manager, or process object is represented by a file. Because these object names are not necessarily valid file names, the queue manager converts the object name into a valid file name, where necessary. This is described in “Understanding MQSeries file names” on page 42.

To find out how to display the real file name of an object, see “dspmqfls (Display MQSeries files)” on page 187.

Stopping a queue manager

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

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

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

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

If you have problems

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

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

Immediate and preemptive queue manager shutdowns

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

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

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

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

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

Working with queue managers

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 this method still does not work, see “Stopping queue managers manually” on page 275 for an alternative.

For a detailed description of the **endmqm** command and its options, see “endmqm (End queue manager)” on page 192.

Restarting a queue manager

To restart a queue manager, use the command:

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

Making an existing queue manager the default

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

You may need to edit this stanza:

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

See Chapter 13, “Configuration files” on page 139 for information about configuration files.

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

Deleting a queue manager

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

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

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

For a description of the **dltmqm** command and its options, see “dltmqm (Delete queue manager)” on page 180. 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 276 for an alternative.

Managing the command server for remote administration

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

Note: For remote administration, you must ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. 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.

Starting the command server

To start the command server use this command:

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

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

Displaying the status of the command server

For remote administration, you must ensure that the command server on the target queue manager is running. If it is not running, no remote commands can be processed. Any messages containing commands are queued in the target queue manager's command queue.

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

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

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

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

Stopping a command server

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

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

You can stop the command server in two different ways:

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

Chapter 6. Administering local MQSeries objects

This chapter describes how to administer local MQSeries objects to support 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”
- “Issuing MQSC commands for administration” on page 50
- “Running MQSC commands from text files” on page 53
- “If you have problems with MQSC” on page 56
- “Working with local queues” on page 58
- “Working with alias queues” on page 65
- “Working with model queues” on page 67
- “Managing objects for triggering” on page 68

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.

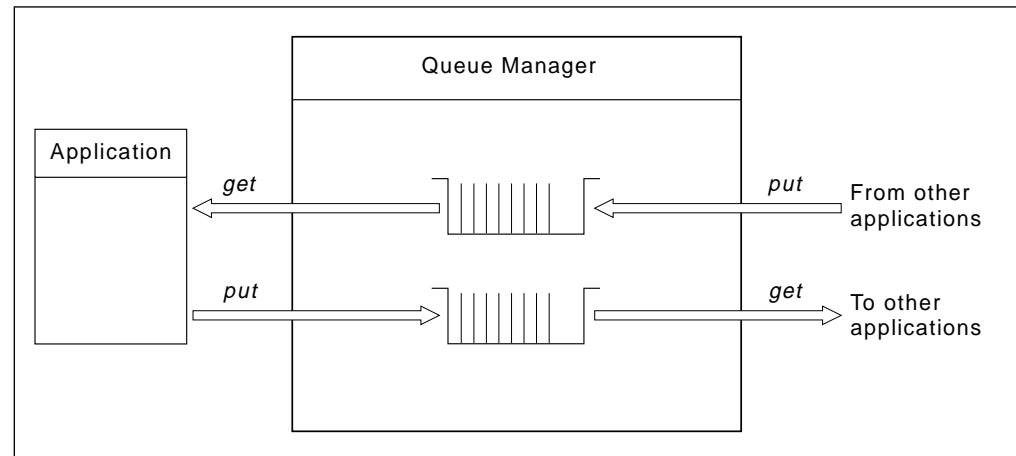


Figure 2. Queues, messages, and applications

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

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

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

Issuing MQSC commands

- The application must be able to connect to the queue manager. To do this it must be linked to the product code. See the *MQSeries Application Programming Guide* for more information.
- The applications that put the messages on the first queue must also connect to a queue manager. If they are remote, they must also be set up with transmission queues and channels. This part of the system is not shown in Figure 2 on page 49.

Issuing MQSC commands for administration

In this section, we assume that you will be issuing commands using the **runmqsc** command. You can do this interactively—entering the commands at the keyboard—or you can redirect SYS\$INPUT to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same.

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

Before you start

Before you can run MQSC commands, you must have created and started the queue manager that is going to run the commands, see “Creating a default queue manager” on page 44.

MQSeries object names

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

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

```
ORANGE.LOCAL.QUEUE
```

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

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

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

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

Case-sensitivity in MQSC commands

MQSeries control commands, for example, **runmqsc** that invokes the MQSC facility is not case-sensitive; see “Using control commands” on page 33.

MQSC commands, including their attributes, can be written in upper or lower case. Object names in MQSC commands are folded (that is, QUEUE and queue are not differentiated), unless the names are put in quotation marks. If quotation marks are not used, the object is processed with a name in uppercase. See the *MQSeries Command Reference* manual for more information.

Redirecting input and output

See “Redirection of SYS\$INPUT, SYS\$OUTPUT, and SYS\$ERROR” on page 24 for details of the redirection indicators that MQSeries for Digital OpenVMS V2.2 supports.

Using the MQSC facility interactively

To enter commands interactively, at a DCL prompt type:

```
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)
```

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:-
```

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

Ending interactive input to MQSC

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

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

Displaying queue manager attributes

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

```
DISPLAY QMGR ALL
```

Issuing MQSC commands

A typical output is:

```
1 : display qmgr all
AMQ8408: Display Queue Manager details.
DESCR( )
DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
DEFXMITQ(MY.DEFAULT.XMIT.QUEUE)
COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
QMNAME("saturn.queue.manager")
TRIGINT(999999999)
MAXHANDS(256)
MAXUMSGS(10000)
AUTHOREV(DISABLED)
INHIBTEV(DISABLED)
LOCALEV(DISABLED)
REMOTEEV(DISABLED)
PERFMEV(DISABLED)
STRSTPEV(ENABLED)
MAXPRTY(9)
CCSID(850)
MAXMSGL(4194304)
CMDLEVEL(100)
PLATFORM(UNIX)
SYNCPT
```

The ALL parameter on the DISPLAY QMGR command causes all the queue manager attributes to be displayed. In particular, because no queue manager name was specified when the command was run, the output tells us the default queue manager name ("saturn.queue.manager"), and the names of the dead-letter queue (SYSTEM.DEAD.LETTER.QUEUE) and the command queue (SYSTEM.ADMIN.COMMAND.QUEUE). Both these queues should have been created when you ran the sample amqscoma.tst; see "Creating the default and system objects" on page 44.

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

You can specify the queue manager name on the **runmqsc** command to run MQSC commands on a local queue manager other than the default. For example, to run MQSC commands on queue manager `jupiter.queue.manager`, 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 76.

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 commands that you want to repeat, over again, you should provide input from a text file. (See “Redirection of `SYS$INPUT`, `SYS$OUTPUT`, and `SYS$ERROR`” on page 24 for information about redirection indicators.) To do this, first create a text file containing the MQSC commands using your familiar text editor. 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 `SYS$INPUT` and `SYS$OUTPUT` 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 3 on page 54 shows an extract from the MQSC command file `myprog.in` and Figure 4 on page 55 shows the corresponding extract of the output in `myprog.out`.

Running MQSC commands

To redirect SYS\$INPUT and SYS\$OUTPUT 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 3 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. The *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

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

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

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

MQSC reports

The **runmqsc** command returns a *report*, which is sent to SYS\$OUTPUT. 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 4 on page 55. 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 only attempts to process 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.
:
.
.
15 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Figure 4. Extract from the MQSC report file, myprog.out.

Running the supplied MQSC command files

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

amqscoma.tst	Default and system objects.
amqscos0.tst	Definitions of objects used by sample programs.

The files are located in the directory `MQS_EXAMPLES:`.

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

```
runmqsc < MQS_EXAMPLES:amqscoma.tst
```

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

Using runmqsc to verify commands

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

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

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

This report is similar to that shown in Figure 4 on page 55.

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

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

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

If you have problems with MQSC

If you cannot get your MQSC commands to run, 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:

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

```
runmqsc amqscoma.tst

5697-270 (C) Copyright IBM Corp. 1995. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

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

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


```
runmqsc < MQS_EXAMPLES:amqscoma.tst > DKA0:[ZONE]myfile.output
```

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

To do this, look in the configuration file `mqsc.ini`, which by default is located in the `MQS_ROOT:[MQM]` directory. This file contains the names of the queue managers and the name of the default queue manager, if you have one.

- The queue manager should already be started, if it is not, start it; see “Starting a queue manager” on page 44. You get an error message if it is already started.
- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, otherwise you get this error:

```
runmqsc <amqscoma.tst

5697-270 (C) Copyright IBM Corp. 1995. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

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

To correct this type of problem, see “Making an existing queue manager the default” on page 46.

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

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

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

```
DEFINE QLOCAL(Queue1)

%DCL-W-PARMDEL, invalid parameter delimiter - check use of special characters
```

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

```
runmqsc
5697-270 (C) Copyright IBM Corp. 1996. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

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

See also “If you have problems using MQSC remotely” on page 77.

Working with local queues

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

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 59.
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 61.

Defining a dead-letter queue

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

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

A dead-letter queue has no special requirements except that it must be a local queue and its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle.

MQSeries provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see Chapter 10, “The MQSeries dead-letter queue handler” on page 107.

Displaying default object attributes

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

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

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

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

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

This command displays the three specified attributes as follows:

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

CURDEPTH is the current queue depth, that is, the number of messages on the queue. This is a useful attribute to display, because 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, and substitute one or more changes to the attributes of the original. For example:

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

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

Notes:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you 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 58, 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
```



```

** Origin Context
PutApplType   : '6'
PutApplName   : '
PutDate      : '19941124'   PutTime   : '11240694'
ApplOriginData : '
**** Message ****

length - 36 bytes

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

No more messages
MQCLOSE
MQDISC

```

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.

Working with alias queues

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

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

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

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

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

You can do this using the following commands:

```
* This alias is put enabled and get disabled for application ALPHA  
  
DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +  
  TARGQ (YELLOW.QUEUE) +  
  PUT (ENABLED) +  
  GET (DISABLED)  
  
* This alias is put disabled and get enabled for application BETA  
  
DEFINE QALIAS (BETAS.ALIAS.QUEUE) +  
  TARGQ (YELLOW.QUEUE) +  
  PUT (DISABLED) +  
  GET (ENABLED)
```

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

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

Using other commands with queue aliases

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

```

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

DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE) ALL

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

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

* Delete this queue alias, if you can.

DELETE QALIAS (ALPHAS.ALIAS.QUEUE)

```

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

Working with model queues

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

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

Defining a model queue

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

```

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

```

Managing objects for triggering

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 SYSYSTEM.DEFAULT.MODEL.QUEUE default queue.

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

Using other commands with model queues

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

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

DISPLAY QUEUE (GREEN.MODEL.QUEUE) ALL

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

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

* Delete this model queue:

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

Managing objects for triggering

MQSeries provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *MQSeries Application Programming Guide*. This section describes how to set up the required objects to support triggering on MQSeries for Digital OpenVMS.

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.INS.QUEUE, as follows:

```

DEFINE QLOCAL (MOTOR.INS.QUEUE) +
PROCESS (MOTOR.INS.PROC) +
MAXMSGL (2000) +
DEFPSIST (YES) +
INITQ (MOTOR.INS.INT.Q) +
TRIGGER +
TRIGTYPE (DEPTH) +
TRIGDPTH (100)+
TRIGMPRI (5)

```

Where:

QLOCAL (MOTOR.INS.QUEUE)

Specifies the name of the application queue being defined.

PROCESS (MOTOR.INS.PROC)

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 are persistent on this queue.

INITQ (MOTOR.INS.INT.Q)

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.INT.Q for guidance:

```

DEFINE QLOCAL(MOTOR.INS.INT.Q) +
GET (ENABLED) +
NOSHARE +
NOTRIGGER +
MAXMSGL (2000) +
MAXDEPTH (10)

```

Creating a process definition

Use the DEFINE PROCESS command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PROCESS attribute on the application queue MOTOR.INS.QUEUE. The following MQSC command defines the required process, MOTOR.INS.PROC, identified in this example:

```
DEFINE PROCESS (MOTOR.INS.PROC) +
    DESCR ('Insurance request message processing') +
    APPLTYPE (OpenVMS) +
    APPLICID ('DKA0:[MQM.ADMIN.TEST]IRMP01.EXE') +
    USERDATA ('open, close, 235')
```

Where:

MOTOR.INS.PROC

Is the name of the process definition, limited to 15 characters.

DESCR ('Insurance request message processing')

Is the descriptive text of the application program to which the definition relates, following the keyword. 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 quotes.

APPLTYPE (OpenVMS)

Is the type of the application that runs on OpenVMS

APPLICID ('[MQM.ADMIN.TEST]IRMP01.EXE')

Is the name of the application executable program.

USERDATA ('open, close, 235')

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

Displaying your process definition

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

```
DISPLAY PROCESS (MOTOR.INS.PROC) ALL

24 : DISPLAY PROCESS (MOTOR.INS.PROC) ALL
AMQ8407: Display Process details.
DESCR ('Insurance request message processing') +
APPLICID ('DKA0:[MQM.ADMIN.TEST]IRMP01.EXE') +
USERDATA (open, close, 235) +
PROCESS (MOTOR.INS.PROC) +
APPLTYPE (OpenVMS)
```

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

Chapter 7. Administering remote MQSeries objects

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

It contains these sections:

- “Understanding channels and remote queuing”
- “Remote administration” on page 72
- “Creating a local definition of a remote queue” on page 78
- “Using remote queue definitions as aliases” on page 81

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

Understanding channels and remote queuing

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

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

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

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

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

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

“Preparing channels and transmission queues for remote administration” on page 73 shows how to use these definitions to set up remote administration.

Remote administration

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

Remote administration

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

- MQSC commands
- PCF commands

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

In remote administration you send MQSC commands to a remote queue manager—either interactively or from a text file containing the commands. The remote queue manager may be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in different MQSeries environments, including AIX, AS/400, MVS/ESA, and OS/2.

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

Preparing queue managers for remote administration

Figure 5 on page 73 shows the configuration of queue managers and channels that are required for remote administration. `source.queue.manager` is the *source* queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned, if possible. `target.queue.manager` is the destination queue manager, which processes the commands and generates any operator messages.

Note: `source.queue.manager` *must* be the default queue manager. For further information on creating a queue manager, see “`crtmqm (Create queue manager)`” on page 176.

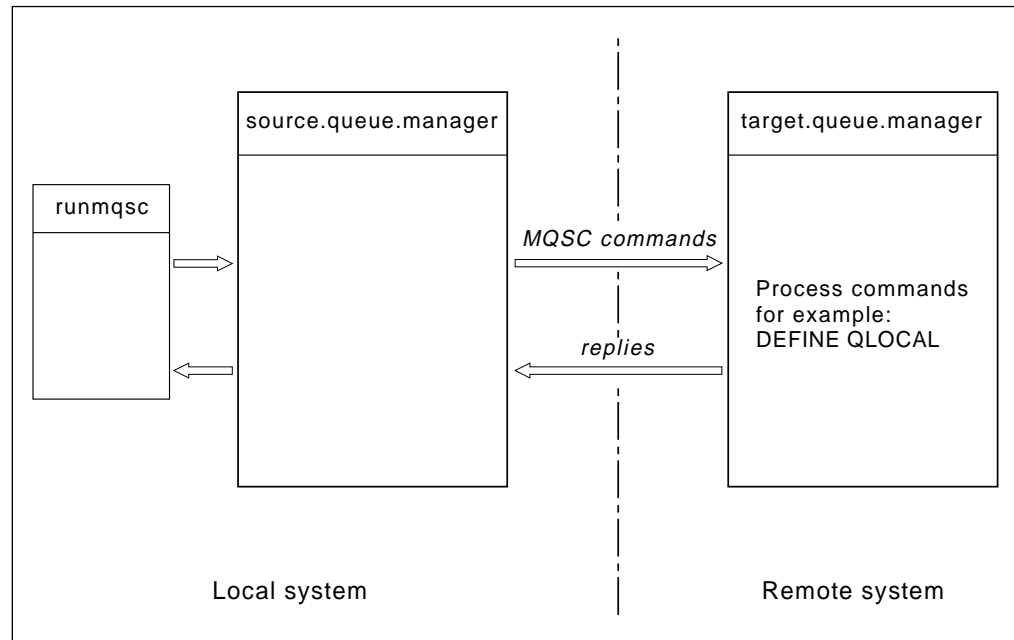


Figure 5. Remote administration

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

- Create the queue manager, using the **crtmqm** command.
- Start the queue manager, using the **strmqm** command.
- Run the sample amqscoma.tst, using the **runmqsc** command.

See “Creating the default and system objects” on page 44 for more information about these steps. You have to run these commands locally or over a network facility, for example Telnet.

On the destination queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This is created from the sample command file amqscoma.tst.
- The command server must be started, using the **strmqcsv** command.

Preparing channels and transmission queues for remote administration

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

The channel source.to.target is for sending MQSC commands from the source queue manager to the destination. Its sender is at source.queue.manager and its receiver is at queue manager target.queue.manager. The channel target.to.source is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each sender. This queue is a local queue that is given the name of the receiving queue manager. Figure 6 on page 74 summarizes this configuration. However, you should be aware that the SYSTEM.MQSC.REPLY.QUEUE is the name of the model queue in

Remote administration

AMQSCOMA.TST that is used by MQSC to develop its own dynamic reply queue. This queue name varies and is internal to MQSC.

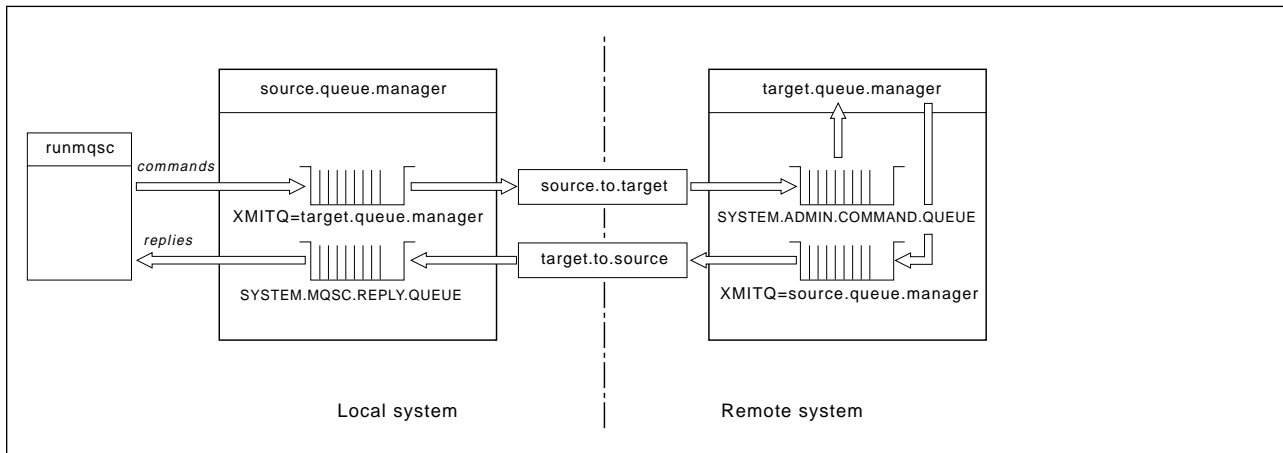


Figure 6. Setting up channels and queues for remote administration

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

Defining channels and transmission queues

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

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

DEFINE CHANNEL ('source.to.target') +
  CHLTYPE(SDR) +
  CONNAME (RH5498) +
  XMITQ ('target.queue.manager') +
  TRPTYPE(TCP)

* Define the receiver channel at the source queue manager

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

* Define the transmission queue on the source

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

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

```

* Define the sender channel on the destination queue manager

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

* Define the receiver channel on the destination queue manager

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

* Define the transmission queue on the destination queue manager

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

```

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

Start the channels

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

To start the two channels, first ensure that the TCP/IP services have been configured for MQSeries on both nodes, and are running at both ends of the connections.

Then start the channels, again as background processes:

- On the source queue manager, type:

```
runmqchl -c "source.to.target" -b
```

- On the destination queue manager, type:

```
runmqchl -c "target.to.source" -b
```

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

Issuing MQSC commands remotely

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

- On the destination queue manager, type:

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

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

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

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

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

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

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

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

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

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

Working with queue managers on MVS/ESA

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

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

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

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

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

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

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

Recommendations for remote queuing

When you are implementing remote queuing:

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, as far as possible, 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 destination queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
 - The channel along which the commands are being sent.
 - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.
- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.

See also “If you have problems with MQSC” on page 56.

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 object 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 destination queue, the destination queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an MQPUT call, specifying the handle returned from the MQOPEN call. The queue manager appends the remote queue name and the remote queue manager name to a transmission header in the message. This information is used to route the message to its correct destination in the network.

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

Example

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 destination queue is owned by another queue manager.

On the MQOPEN call, the application specifies these fields:

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

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

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

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

Where:

QREMOTE (CYAN.REMOTE.QUEUE)

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

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

Additional text that describes the use of the queue.

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)

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

RQMNAME (jupiter.queue.manager)

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

XMITQ (INQUOTE.XMIT.QUEUE)

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

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

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.

Local definition of remote queue

Using other commands with remote queues

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

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

DISPLAY QUEUE (CYAN.REMOTE.QUEUE) ALL

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

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

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

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

Note: If you delete a remote queue, you only delete the local representation of the remote queue. You do not delete the remote queue 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 destination queue manager exists, that queue is used. If this queue does not exist, the queue specified by the *DefaultXmitQ* attribute (DEFXMITQ in MQSC) on the local queue manager is used.

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

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

Applications can put messages directly on a transmission queue, with an appropriate header, 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 78.

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

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

Queue manager aliases

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

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

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

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.

Aliases

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

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

Chapter 8. Security

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

It contains these sections:

- “Before you begin”
- “Why you need to protect MQSeries resources” on page 84
- “Understanding the Object Authority Manager” on page 84
- “Using the Object Authority Manager commands” on page 87
- “Object Authority Manager guidelines” on page 89
- “Understanding the authorization specification tables” on page 93
- “Understanding authorization files” on page 98

Before you begin

All queue manager resources run with the VMS Rights Identifier:

MQM

This rights identifier is created during MQSeries installation and you **must** grant this resource attribute to all users who need to control MQSeries resources.

User IDs in MQSeries for Digital OpenVMS with resource identifier MQM

If your user ID holds the MQM OpenVMS rights identifier, you have all authorities to all MQSeries resources. Your user ID **must** hold the OpenVMS MQM rights identifier to be able to use all the MQSeries for Digital OpenVMS control commands except **crtmqcvx**. In particular, you need this authority to:

- Use the **runmqsc** command to run MQSC commands.
- Administer authorities on MQSeries for Digital OpenVMS using the **setmqaut** command.

If you are sending channel commands to queue managers on a remote system, you must ensure that your user ID holds the OpenVMS rights identifier MQM on the target system. For a list of PCF and MQSC channel commands, see “Channel command security” on page 92.

It is not essential for your user ID to hold the rights identifier MQM for issuing:

- PCF commands—including Escape PCFs—from an administration program
- MQI calls from an application program

For more information

For more information about:

- MQSeries for Digital OpenVMS command sets, see Chapter 4, “Understanding administration command sets” on page 33.
- MQSeries for Digital OpenVMS control commands, see Chapter 15, “MQSeries control commands” on page 171.

Object authority manager

- PCF commands and Escape PCFs, see the *MQSeries Programmable System Management* manual.
- MQI calls, see the *MQSeries Application Programming Guide* and *MQSeries Application Programming Reference* manuals.

Why you need to protect MQSeries resources

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

- Connections to a queue manager.
- Access to MQSeries objects such as queues, channels, and processes.
- Commands for queue manager administration, including MQSC commands and PCF commands.
- Access to MQSeries messages.
- Context information associated with messages.

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

Understanding the Object Authority Manager

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

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

- Replace the supplied OAM with your own authorization service component using the interface provided.
- Augment the facilities supplied by the OAM with those of your own authorization service component, again using the interface provided.
- Remove or disable the OAM and run with no authorization service at all.

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

The OAM manages users' authorizations to manipulate MQSeries objects, including queues, process definitions, and channels. 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.

How the OAM works

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

Managing access through rights identifiers

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

Rights identifiers and the primary rights identifier

Managing access permissions to MQSeries resources is based on OpenVMS *rights identifiers*, that is, identifiers held by principals. A principal can hold one or more OpenVMS rights identifiers. A group is defined as the set of all principals that have been granted a specific rights identifier.

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

When a principal holds more than one rights identifier

The authorizations that a principal has are the union of the authorizations of all the rights identifiers that it holds, that is, its process rights. Whenever a principal requests access to a resource, the OAM computes this union, and then checks the authorization against it. You can use the control command **setmqaut** to set the authorizations for a specific principal, or identifier.

Note: Any changes made using the **setmqaut** command take immediate effect, unless the object is in use. In this case, the change comes into force when the object is next opened. However, changes to a principal's rights identifier list do not come into effect until a queue manager is reset, that is, stopped and restarted.

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

Default rights identifier

The OAM recognizes a default to which all users are nominally assigned. This group is defined by the pseudo rights identifier of 'NOBODY'. 'NOBODY' can be used as if it were a valid rights identifier to assign authorizations using MQSeries commands. By default, no authorizations are given to this identifier. Users without specific authorizations can be granted access to MQSeries resources through this rights identifier.

Resources you can protect with the OAM

Through OAM you can control:

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

In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use MQSC commands; only principals which hold rights identifier mqm can execute queue manager administration commands, for example, to create a queue.
- Permission to use control commands; only principals which hold rights identifier mqm can execute control commands, for example, creating a queue manager, starting a command server, or using **runmqsc**.
- Permission to use PCF commands.

Different users may be granted different kinds of access authority to the same object. For example, for a specific queue, users holding one identifier may be allowed to perform both put and get operations; users with another identifier may only be allowed to browse the queue (MQGET with browse option). Similarly, users with identifiers may have get and put authority to a queue, but are not allowed to alter or delete the queue.

Using rights identifiers for authorizations

Using identifiers, 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 an identifier consisting of end users who want to run a particular application. New users can be given access simply by granting the appropriate identifier to their OpenVMS user ID.

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

Disabling the object authority manager

By default, the OAM is enabled. You can disable it by setting the logical name MQSNOAUT before the queue manager is created, as follows:

```
$ DEFINE/SYSTEM MQSNOAUT TRUE
```

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

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

Using the 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 – your user ID must hold the OpenVMS rights identifier MQM. This identifier should have been set up when you installed the product.

If your user ID holds identifier mqm, you have a ‘super user’ authority to the queue manager. This means that you are authorized to issue any MQI request or command from your user ID.

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

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

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

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

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

What you specify when you use the OAM commands

The authority commands apply to the specified queue manager; if you do not specify a queue manager, the default queue manager is used. On these commands, you must specify the object uniquely, that is, you must specify the object name and its type. You also have to specify the principal or identifier name to which the authority applies.

Authorization lists

On the **setmqaut** command you specify a list of authorizations. This is simply a shorthand way of specifying whether authorization is to be granted or revoked, and which resources the authorization applies to. Each authorization in the list is specified as a lowercase keyword, prefixed with a + or – sign. Use a + sign to add the specified authorization or a – sign to remove the authorization. You can specify any number of authorizations in a single command. For example:

```
+browse -get +put
```

Using the setmqaut command

Provided you have the required authorization, you can use the **setmqaut** command to grant or revoke authorization of a principal or rights identifier to access a particular object. The following example shows how the **setmqaut** command is used:

Using OAM commands

```
setmqaut -m "saturn.queue.manager" -t queue -n RED.LOCAL.QUEUE -g GROUPA +browse -get +put
```

In this example:

This term...	Specifies the...
saturn.queue.manager	Queue manager name.
queue	Object type.
RED.LOCAL.QUEUE	Object name.
GROUPA	ID of the group to be given the authorizations.
+browse -get +put	Authorization list for the specified queue. There must be no spaces between the '+' or '-' signs and the keyword.

The authorization list specifies the authorizations to be given, where:

This term...	Does this...
+browse	Adds authorization to browse (MQGET with browse option) messages on the queue.
-get	Removes authorization to get (MQGET) messages from the queue.
+put	Adds authorization to put (MQPUT) messages on the queue.

This means that applications started with user IDs that hold OpenVMS identifier GROUPA have these authorizations.

You can specify one or more principals and, at the same time, one or more identifiers. For example, the following command revokes put authority on the queue MyQueue to the principal FVUSER and to identifiers GROUPA and groupb.

```
setmqaut -m "saturn.queue.manager" -t queue -n "MyQueue" -p FVUSER -g GROUPA -g GROUPB -put
```

Note: This command also revokes put authority for all rights identifiers held by FVUSER, that is, all groups to which FVUSER belongs.

For a formal definition of the command and its syntax, see "setmqaut (Set/reset authority)" on page 211.

Authority commands and installable services

The **setmqaut** command takes an additional parameter that specifies the name of the installable service component to which the update applies. You must specify this parameter if you have multiple installable components running at the same time. By default, this is not the case. If the parameter is omitted, the update is made to the first installable service of that type, if one exists. By default, this is the supplied OAM.

Access authorizations

Authorizations defined by the authorization list associated with the **setmqaut** command can be categorized as follows:

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

Each authorization is specified by a keyword used with the **setmqaut** and **dspmqaut** commands. These are described in “setmqaut (Set/reset authority)” on page 211.

Display authority command

You can use the command **dspmqaut** to view the authorizations that a specific principal or identifier has for a particular object. The flags have the same meaning as those in the **setmqaut** command. Authorization can only be displayed for one identifier or principal at a time. See “dspmqaut (Display authority)” on page 182 for a formal specification of this command.

For example, the following command displays the authorizations that the group GpAdmin has to a process definition named Annuities on queue manager QueueMan1.

```
dspmqaut -m "QueueMan1" -t process -n "Annuities" -g GPADMIN
```

The keywords displayed as a result of this command identify the authorizations that are active.

Object Authority Manager guidelines

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

- Starting and stopping queue managers.
- Accessing certain special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE.
- Programs that use full MQI context options.
- In general, creating and copying application queues.

User IDs

The special user ID MQM that you created during product installation is intended for use by the product only. It should never be available to non-privileged users.

The user ID used for authorization checks, associated with an MQ process, is the OpenVMS user ID.

Queue manager directories

The directory containing queues and other queue manager data is private to the product. Objects in this directory have OpenVMS user authorizations that relate to their OAM authorizations. However, do not use standard OpenVMS commands to grant or revoke authorizations to MQI resources because:

- MQSeries objects are not necessarily the same as the corresponding system object name. See “Understanding MQSeries file names” on page 42 for more information about this.
- All objects are owned by resource ID mqm.

Queues

The authority to a dynamic queue is based on—but not necessarily the same as—that of the model queue from which it is derived. See page 95 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, some users may bypass the normal access control simply by creating an alias.

Alternate user authority

Alternate user authority controls whether one user ID can use the authority of another user ID when accessing an MQSeries object. This is essential where a server receives requests from a program and the server wishes to ensure that the program has the required authority for the request. The server may have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, you can use alternate user authority to control whether PAYSERV is allowed to specify USER1 as an alternate user ID when it opens the reply-to queue.

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

Note: You can use alternate user IDs on any MQSeries object. Use of an alternate user ID does not affect the user ID used by any other resource managers.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. The context information comes in two sections:

Identity section This part specifies who the message came from. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section This section specifies where the message came from, and when it was put onto the queue. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQOPEN or an MQPUT call is made. This data may be generated by the application, it may be passed on from another message, or it may be generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application, running under an authorized user ID.

A server program can use the *UserIdentifier* to determine the user ID of an alternate user.

You use context authorization to control whether the user can specify any of the context options on any MQOPEN or MQPUT1 call. For information about the context options, see the *MQSeries Application Programming Guide*. For descriptions of the message descriptor fields relating to context, see the *MQSeries Application Programming Reference* manual.

Remote security considerations

For remote security, you should consider:

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

Specify the channel attribute PUTAUT as follows:

DEF Default user ID. This is the user ID that the message channel agent is running under.

CTX The user ID in the message context.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required for this. However, putting a message directly on a transmission queue requires special authorization; see Table 7 on page 94.

Channel exits Channel exits can be used for added security.

For more information, see the *MQSeries Distributed Queuing Guide*.

Channel command security

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

PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote OpenVMS system. The user ID, as specified in the message descriptor of the PCF message, must hold rights identifier mqm on the target system. These commands are:

- *ChangeChannel*
- *CopyChannel*
- *CreateChannel*
- *DeleteChannel*
- *PingChannel*
- *ResetChannel*
- *StartChannel*
- *StartChannelInitiator*
- *StopChannel*
- *ResolveChannel*

See the *MQSeries Programmable System Management* manual for the PCF security requirements.

MQSC channel commands

You can issue MQSC channel commands to a remote OpenVMS 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 hold rights identifier mqm on the target system. (PCF commands are implicit in MQSC commands issued from **runmqsc** in indirect mode.) These commands are:

- ALTER CHANNEL
- DEFINE CHANNEL
- DELETE CHANNEL
- PING CHANNEL
- RESET CHANNEL
- START CHANNEL
- START CHINIT
- STOP CHANNEL
- RESOLVE CHANNEL

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

Control commands for channels

For the control commands for channels, the user ID that issues them must hold rights identifier mqm. These commands are:

- **runmqchi** (Run channel initiator)
- **runmqchl** (Run channel)

Understanding the authorization specification tables

The authorization specification tables starting on page 94 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 “What the authorization files contain” on page 99 for more information.

MQI authorizations

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

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

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

Table 7 on page 94 summarizes the authorizations needed for each call.

Authorization specification tables

<i>Table 7. Security authorization needed for MQI calls</i>			
Authorization required for :	Queue object (1)	Process object	Queue manager object
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT
MQOPEN Option			
MQOO_INQUIRE	MQZAO_INQUIRE (2)	MQZAO_INQUIRE (2)	MQZAO_INQUIRE (2)
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT (3)	MQZAO_INPUT	Not applicable	No check
MQOO_OUTPUT (Normal queue) (4)	MQZAO_OUTPUT	Not applicable	No check
MQOO_PASS_ IDENTITY_CONTEXT (5)	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ ALL_CONTEXT (5, 6)	MQZAO_PASS_ ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT (5, 6)	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT (7)
MQOO_SET_ ALL_CONTEXT (5, 8)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQOO_OUTPUT (Transmission queue) (9)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	(10)	(10)	MQZAO_ALTERNATE_ USER_AUTHORITY (10, 11)
MQPUT1 Option			
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT (12)	Not applicable	No check
MQPMO_PASS_ ALL_CONTEXT	MQZAO_PASS_ ALL_CONTEXT (12)	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT (12)	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT (7)
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT (12)	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
(Transmission queue) (9)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQPMO_ALTERNATE_ USER_AUTHORITY	(13)	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY (11)
MQCLOSE Option			
MQCO_DELETE	MQZAO_DELETE (14)	Not applicable	Not applicable
MQCO_DELETE_PURGE	MQZAO_DELETE (14)	Not applicable	Not applicable

Specific notes:

1. If a model queue is being opened:
 - MQZAO_DISPLAY authority is needed for the model queue, in addition to whatever other authorities (also for the model queue) are required for the open options specified.
 - MQZAO_CREATE authority is not needed to create the dynamic queue.
 - The user identifier used to open the model queue is automatically granted all of the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. Either the queue, process, 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.

Authorization specification tables

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 on page 95) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. 'No check' means that no authorization checking is carried out.
4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Administration authorizations

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

Authorizations for MQSC commands in escape PCFs

Table 8 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

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

Specific notes:

1. The user identifier, under which the program (for example, **runmqsc**) which submits the command is running, must also have MQZAO_CONNECT authority to the queue manager.
2. Either the queue, process, or queue manager object is checked, depending on the type of object.

3. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. This applies if the object to be replaced does in fact already exist. If it does not, the check is as for DEFINE object NOREPLACE.

General notes:

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

Authorizations for PCF commands

Table 9 summarizes the authorizations needed for each PCF command.

<i>Table 9. PCF commands and security authorization needed</i>			
(2) Authorization required for:	Queue object	Process object	Queue manager object
PCF command			
Change object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
Clear Queue	MQZAO_CLEAR	Not applicable	Not applicable
Copy object (without replace) (3)	MQZAO_CREATE (4)	MQZAO_CREATE (4)	Not applicable
Copy object (with replace) (3, 6)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable
Create object (without replace) (5)	MQZAO_CREATE (4)	MQZAO_CREATE (4)	Not applicable
Create object (with replace) (5, 6)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable
Delete object	MQZAO_DELETE	MQZAO_DELETE	Not applicable
Inquire object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY
Inquire object names	No check	No check	No check
Reset queue statistics	MQZAO_DISPLAY and MQZAO_CHANGE	Not applicable	Not applicable

Specific notes:

1. The user identifier under which the program submitting the command is running must also have authority to connect to its local queue manager, and to open the command admin queue for output.
2. Either the queue, process, or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
4. The MQZAO_CREATE authority is not specific to a particular object 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.

Authorization files

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_DISPLAYMQZAO_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.

Understanding authorization files

Note: The information in this section is given for problem determination. Under normal circumstances, use authorization commands to view and change authorization information.

MQSeries for Digital OpenVMS uses a specific file structure to implement security. You do not have to do anything with these files, except to ensure that all the authorization files are themselves secure.

Security is implemented by authorization files. From this perspective, there are three types of authorization:

- Authorizations applying to single object, for example, the authority to put a message on an queue.
- Authorizations applying to a class of objects, for example, the authority to create a queue.
- Authorizations applying across all classes of objects, for example, the authority to perform operations on behalf of different 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 sub-directory, the path of which is defined by the queue manager name, the type of authorization, and where appropriate, the object name.

Not all authorizations apply directly to instances of objects. For example, the authorization to create an object applies to the class of objects rather than to an

individual instance. Also, some authorizations apply across the entire queue manager, for example, alternate user authority means that a user can assume the authorities associated with another user.

Authorization directories

By default, the authorization directories, for a queue manager called saturn are:

MQS_ROOT: [MQM.QMGRS.SATURN.AUTH.QUEUES]

Authorization files for queues.

MQS_ROOT: [MQM.QMGRS.SATURN.AUTH.PROCDEF]

Authorization files for process definitions.

MQS_ROOT: [MQM.QMGRS.SATURN.AUTH.QMANAGER]

Authorization files for the queue manager.

MQS_ROOT: [MQM.QMGRS.SATURN.AUTH]\$ACCLASS

Authorizations applying to all classes.

In the object directories, the \$CLASS files hold the authorizations related to the entire class.

Note: There is a difference between \$CLASS (the authorization file that specifies authorization for a particular class) and \$ACCLASS (the directory that contains a file that specifies authorizations to all classes)

The paths of the object authorization files are based on those of the object itself, where auth is inserted ahead of the object type directory. You can use the **dspmqls** command to display the path to a specified object.

For example, if the name and path of SYSTEM.DEFAULT.LOCAL.QUEUE is:

MQS_ROOT: [MQM.QMGRS.SATURN.QUEUES.SYSTEM\$DEFAULT\$LOCAL\$QUEUE]

the name and path of the corresponding authorization file is:

MQS_ROOT: [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 42 for details.

What the authorization files contain

The authorizations of a particular identifier are defined by a set of stanzas in the authorization file. See “Understanding authorization files” on page 98 for more information. The authorizations apply to the object associated with this file. For example:

```
groupb:
  Authority=0x0040007
```

This stanza defines the authority for the identifier groupb. The authority specification is the union of the individual bit patterns based on the following assignments:

Authorization files

Authorization keyword	Formal name	Hexadecimal Value
connect	MQZAO_CONNECT	0x00000001
browse	MQZAO_BROWSE	0x00000002
get	MQZAO_INPUT	0x00000004
put	MQZAO_OUTPUT	0x00000008
inq	MQZAO_INQUIRE	0x00000010
set	MQZAO_SET	0x00000020
passid	MQZAO_PASS_IDENTITY_CONTEXT	0x00000040
passall	MQZAO_PASS_ALL_CONTEXT	0x00000080
setid	MQZAO_SET_IDENTITY_CONTEXT	0x00000100
setall	MQZAO_SET_ALL_CONTEXT	0x00000200
altusr	MQZAO_ALTERNATE_USER_AUTHORITY	0x00000400
allmqi	MQZAO_ALL_MQI	0x000007FF
crt	MQZAO_CREATE	0x00010000
dlt	MQZAO_DELETE	0x00020000
dsp	MQZAO_DISPLAY	0x00040000
chg	MQZAO_CHANGE	0x00080000
clr	MQZAO_CLEAR	0x00100000
chgaut	MQZAO_AUTHORIZE	0x00800000
alladm	MQZAO_ALL_ADMIN	0x009E0000
none	MQZAO_NONE	0x00000000
all	MQZAO_ALL	0x009E07FF

These definitions are made in the header file cmqzc.h. In the following example, groupb has been granted authorizations based on the hexadecimal number 0x40007. This corresponds to:

MQZAO_CONNECT	0x00000001
MQZAO_BROWSE	0x00000002
MQZAO_INPUT	0x00000004
MQZAO_DISPLAY	0x00040000

Authority is:	0x00040007

These access rights mean that anyone in groupb can issue the MQI calls:

```
MQCONN
MQGET (with browse)
MQPUT
```

They also have DISPLAY authority for the object associated with this authorization file.

Class authorization files

The *class authorization files* hold authorizations that relate to the entire class. These files are called "\$CLASS" and exist in the same directory as the files for specific objects. The entry MQZAO_CRT in the \$CLASS file gives authorization to create an object in the class. This is the only class authority.

All class authorization files

The *all class authorization file* holds authorizations that apply to an entire queue manager. This file is called \$ACCLASS 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...

Gives authorization to...

MQZAO_ALTERNATE_USER_AUTHORITY

Assume the identity of another user when interacting with MQSeries objects.

MQZAO_SET_ALL_CONTEXT

Set the context of a message when issuing MQPUT.

MQZAO_SET_IDENTITY_CONTEXT

Set the identity context of a message when issuing MQPUT.

Managing authorization files

Here are some pointers that you need to take into consideration when managing your authorization files:

1. You must ensure that the authorization files are secure and not write-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:
 - Backup the auth subdirectory after any significant updates
 - Retain DCL 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

Authorization files must be readable by any principal. However, only the system manager and user with the mqm identifier should be allowed to update these files.

The permissions on authorization files, created by the OAM, are:

```
S:RWED,W:R      (ID=MQM,ACCESS=R+W+D)
```

Do not alter these permissions without reviewing carefully whether there are any security exposures.

To alter authorizations using the command supplied with MQSeries for Digital OpenVMS, your process must have the MQM rights identifier.

Authorization files

Chapter 9. Using the name service

The name service is an installable service that enables an application connected to one queue manager to open what it thinks are local queues. These queues are actually queues defined on another queue manager – on another machine – with the SCOPE attribute set to CELL.

The application can perform all the operations permitted for remote queues on queues opened in this way. The supplied implementation uses DCE (Distributed Computing Environment), although you are free to write your own component that does not use DCE.

To use the supplied name service component, you must define the name service and its installed component to the queue manager. You do this by inserting the appropriate stanza in the queue manager configuration file (qm.ini) file. See the *MQSeries Programmable System Management* manual for details. You will also need to do some DCE configuration.

Using DCE to share queues on different queue managers

If your queue managers are located on nodes within a Distributed Computing Environment (DCE) cell, you can configure them to share queues. Applications can then connect to one queue manager and open a queue on *another* queue manager on another node. To the application, this is transparent; it is not aware that the queue actually resides on another queue manager. (Normally, the queue manager rejects open requests from a local application if the queue does not exist on that queue manager.)

Configuration tasks for shared queues

This section describes how you set up shared queues on queue managers that reside on nodes that are within the DCE cell.

For each queue manager:

1. Configure the name service by adding the required name service stanza to the queue manager configuration file. The contents of this stanza are described in *MQSeries Programmable System Management*. To invoke the name service, you have to restart the queue manager.
2. Use the **endmqm** command to stop the queue manager if it is running.
3. Use the **strmqm** to restart the queue manager.
4. Set up channels for messaging between queue managers; see “Preparing channels and transmission queues for remote administration” on page 73.

For any queue that you want to be shared, specify the SCOPE attribute as CELL. For example, use these MQSC commands:

```
DEFINE QLOCAL (GREY.PUBLIC.QUEUE) SCOPE(CELL)
or
ALTER QLOCAL (PINK.LOCAL.QUEUE) SCOPE(CELL)
```

DCE configuration

The queue created or altered must belong to a queue manager on a node within the DCE cell.

DCE configuration

To use the supplied name service component, you must have the OSF Distributed Computing Environment (DCE) installed. This service enables applications that connect to one queue manager to open queues that belong to another queue manager in the same DCE cell.

The following DCL shell script sets up the DCE namespace so that the supplied name service can run.

```
#!/*****  
$!/* Module Name: setup.com */  
$!/* */  
$!/* Module Type: Command File */  
$!/* */  
$!/* Function : Set up the DCE namespace so that the DCE Name Service */  
$!/* can run */  
$!/* */  
$!/* Version : 2.0 */  
$!/* Automatic binding using the namespace */  
$!/* */  
$!/* Usage : Run this script */  
$!/* $ @setup.com */  
$!/* */  
$!/* Source : New code */  
$!/* */  
$!/* Notes : */  
$!/* 1) You must be logged into VMS as the user who will run the DCE */  
$!/* Naming service */  
$!/* 2) You must be logged into DCE as cell_admin to run this */  
$!/* 3) This needs to be done only once */  
$!/* (c) Copyright International Business Machines Corporation 1997 */  
#!/*****  
$!  
$ ECHO := WRITE SYS$OUTPUT  
$ MKDIR := CREATE/DIRECTORY  
$ RM := DELETE/NOLOG  
$!  
$ PRINC := mqm  
$ PASSW := pwd  
$ MYPASSW := mypwd  
$ DIRECTORY1 := MQSeries  
$ DIRECTORY2 := NamingService  
$!  
#!/*****  
$!/* Set up the queue manager prefix - should come from mqs.ini */  
#!/*****  
$!  
$ PREFIX := MQS_ROOT:[MQM]  
$!  
#!/*****  
$!/* Check that the prefix directory already exists */  
$!/* If it does, get the name of the DCE directory, the name of the */  
$!/* DCE keytab directory & the name of the keytab file */  
#!/*****  
$!  
$!  
$ if f$parse (PREFIX) .eqs ""  
$ then  
$ echo "ERROR: The directory 'PREFIX' must already exist"  
$ exit  
$ else  
$ MQMDCE = PREFIX - "]" + ".dce]"
```



```

$      DCEKEYTAB      = MQMDCE - "]" + ".keytabs]"
$      KEYTAB         = DCEKEYTAB + "keytab"
$      endif
$!
$!
$!/*****/
$!/* If DCE directory does not exist, create it */
$!/* otherwise If DCE keytab directory does not exist, create it */
$!/* otherwise make sure the keytab file does not exist */
$!/*****/
$!
$!
$      if f$parse(MQMDCE) .eqs. ""
$      then
$          echo "Creating the directory 'MQMDCE'"
$          mkdir 'MQMDCE'
$      endif
$!
$!
$      if f$parse(dcekeytab) .eqs. ""
$      then
$          echo "Creating the directory 'DCEKEYTAB'"
$          mkdir 'DCEKEYTAB'
$      else
$          if f$search(keytab) .nes. ""
$          then
$              echo "Deleting the file 'KEYTAB'"
$              rm 'KEYTAB';*
$          endif
$      endif
$!
$!/*****/
$!/* Add entries to define the principal and account for the server */
$!/* Then we add the password to the private keytab file. Access to the */
$!/* keytab file lets the server retrieve the password for the account */
$!/* and authenticate with it. */
$!/*****/
$!
$!
$      @sys$manager:dce$define_required_commands
$!
$      open/write outf rgy_cfg.com
$      write outf "$ rgy_edit"
$      write outf " domain principal"
$      write outf " add 'PRINC'"
$      write outf " domain account"
$      write outf " add 'PRINC' -g none -o none -pw ""'PASSW'"" -mp""'MYPASSW'""
$      write outf " ktadd -p 'PRINC' -f 'KEYTAB' -pw 'PASSW'"
$      write outf " quit"
$      write outf "$ exit"
$      close outf
$!
$      @rgy_cfg.com
$      rm rgy_cfg.com;*
$      echo "Principal and account added & the keytab file created"
$!
$!
$!/*****/
$!/* Change the keytab file permissions so MQM can read it */
$!/*****/
$!
$!
$      if f$search(KEYTAB) .nes. ""
$      then
$          set file 'KEYTAB' /prot=(o:r,g:r)
$          set file 'KEYTAB' /own=mqm
$          echo "Keytab file permission modes changed"
$      endif
$!
$!/*****/
$!/* Build the directories to hold the server entries */
$!/*****/

```

DCE configuration

```
$!  
$!  
$ open/write outf cds_cfg.com  
$ write outf "$ cdscp create dir ./:/subsys/'DIRECTORY1'"  
$ write outf "$ cdscp create dir ./:/subsys/'DIRECTORY1'/'DIRECTORY2'"  
$ close outf  
$!  
$ @cds_cfg.com  
$ rm cds_cfg.com;*  
$ echo "CDS directories for server entries complete"  
$!  
$!/*****/  
$!/* Give the account some privileges so it can access the directories */  
$!/*****/  
$!  
$!  
$ open/write outf acl_cfg.com  
$ write outf "$ acl_edit ./:/ -m user:''PRINC':rwdtcia"  
$ write outf "$ acl_edit ./:/subsys -m user:''PRINC':rwdtcia"  
$ write outf "$ acl_edit ./:/subsys/' DIRECTORY1' -m user:''PRINC':rwdtcia"  
$ write outf "$ acl_edit ./:/subsys/' DIRECTORY1'/'DIRECTORY2' -m user:''PRINC':rwdtcia"  
$ close outf  
$!  
$ @acl_cfg.com  
$ rm acl_cfg.com;*  
$ echo "Account given privileges to access directories"  
$!  
$ exit
```

Chapter 10. The MQSeries dead-letter queue handler

A *dead-letter queue* (DLQ), sometimes referred to as an *undelivered-message queue*, is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have an associated DLQ.¹

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”
- “The DLQ handler rules table” on page 108
- “How the rules table is processed” on page 115
- “An example DLQ handler rules table” on page 117

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 < rule.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.

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

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 SYS\$INPUT: you associate the rules table with **runmqdlq** by redirecting SYS\$INPUT from the rules table.

Attention Running the DLQ handler without redirecting SYS\$INPUT to a rule file causes the DLQ handler to loop.

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 203.

The sample DLQ handler, amqsdq

In addition to the DLQ handler invoked using the **runmqdlq** command, MQSeries provides the source of a sample DLQ handler, amqsdq, whose function is similar to that provided via **runmqdlq**. You can customize amqsdq 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, amqsdq, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

The source of amqsdq is supplied in the directory:

[.DLQ], under MQS_EXAMPLES

and the compiled version is supplied in the directory:

[.BIN], under MQS_EXAMPLES

The DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table. Please note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives, 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 reinvoke it by means of triggering.

As an alternative to including control data in the rules table, you can supply the names of the DLQ and its queue manager as input parameters of the **runmqdlq** command. If any value is specified both in the rules table and on input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

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

Rules (patterns and actions)

Figure 7 shows an example rule from a DLQ handler rules table.

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

Figure 7. 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. Please note the following:

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

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

The pattern-matching keywords

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

APPLIDAT (*ApplIdentityData**)

Is the *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutAppName**)

Is the name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutAppName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutAppType**)

Is the *PutAppType* 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.
- 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 Appendix N, "Messages" on page 309.) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
6. Multiple instances of the DLQ handler could run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ will be seen, even if the DLQ is defined as first-in-first-out (FIFO). Therefore, if the queue is not empty, a periodic rescan of the DLQ is performed to check all messages. For these reasons, you should try to ensure that the DLQ contains as few messages as possible; if messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which simply leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, the final rule in the table should be a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This action causes messages that fall through to the final rule in the table to be forwarded to the queue REALLY.DEAD.QUEUE, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table

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.
```

DLQ handler

```
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 11. Instrumentation events

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

What instrumentation events are

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

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

Other conditions that can give rise to instrumentation events include:

- A threshold limit for the number of messages on a queue being reached.
- A queue not being serviced within a specified time period.
- A channel instance being started or stopped.
- In MQSeries for an OpenVMS system, an application attempting to open a queue specifying a user ID that is not authorized.

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

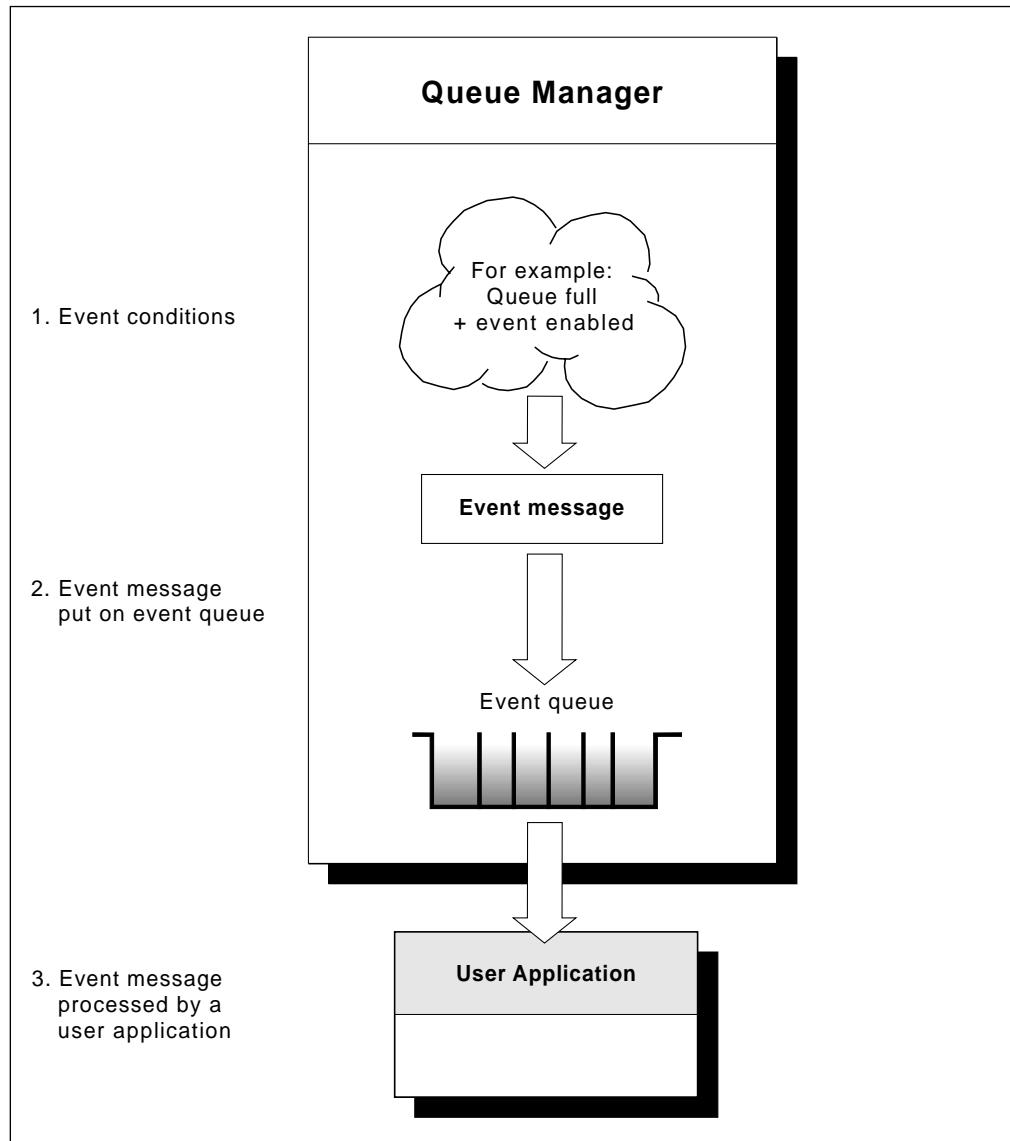


Figure 8. Understanding instrumentation events. When a queue manager detects that the conditions for an event have been met, it puts an event message on the appropriate event queue.

The event message, which contains information about the conditions giving rise to the event, is put onto an *event queue*. An application can retrieve the event message from this queue for analysis.

Why use events?

If you specify your event queues as 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 9 on page 121 illustrates this.

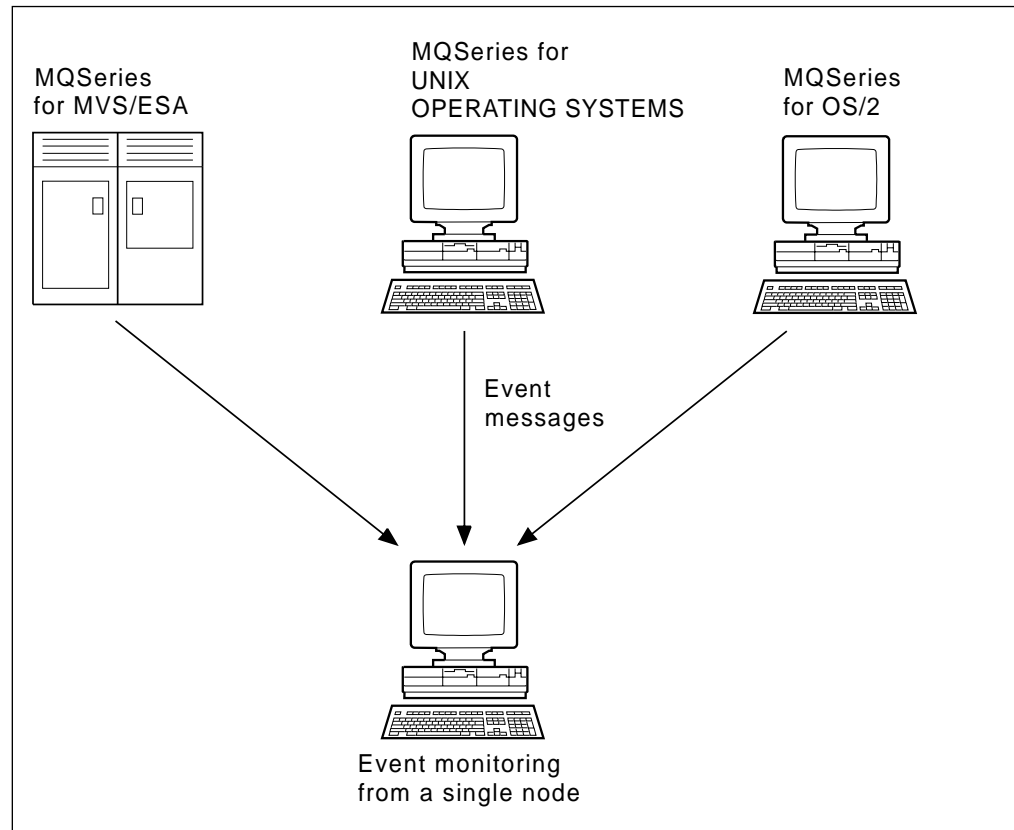


Figure 9. Monitoring queue managers across different platforms, on a single node

Types of events

MQSeries events may be categorized as follows:

Queue manager events

These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events

These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached or, following a get, the queue was not serviced within a predefined time limit.

Channel events

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

Trigger events

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

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

Event notification through event queues

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

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

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

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

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

Using triggered event queues

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

Enabling and disabling events

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

- MQSC commands. For more information, see the *MQSeries Command Reference* manual.
- PCF commands for queue managers on UNIX systems, OpenVMS systems, and OS/2. For more information, see the *MQSeries Programmable System Management* manual.

Enabling an event depends on the category of the event:

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

Event messages

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

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

Chapter 12. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. This means that it must provide a method of tracking the messages in the system, and of recovering messages if the system fails for any reason.

MQSeries ensures that messages are not lost by maintaining records (logs) of the activities of the queue managers that handle the receipt, transmission, and delivery of messages. It uses these logs for three types of recovery:

1. *Restart recovery*, when you stop MQSeries in a planned way.
2. *Crash recovery*, when MQSeries is stopped by an unexpected failure.
3. *Media recovery*, to restore damaged objects.

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped. Any in-flight transactions are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; non-persistent messages are lost during the process.

The rest of this chapter introduces the concepts of recovery and restart in more detail and then tells you how to recover if problems occur. It covers the following topics:

- “Making sure that messages are not lost (logging)”
- “Checkpointing – ensuring complete recovery” on page 128
- “Managing logs” on page 130
- “Using the log for recovery” on page 132
- “Backup and restore” on page 134
- “Recovery scenarios” on page 136

Making sure that messages are not lost (logging)

MQSeries records all significant changes to the data controlled by the queue manager in a log. This includes the creation and deletion of objects, all persistent message updates, transaction states, changes to object attributes, and channel activities. Therefore, the log contains the information you need to recover all updates to message queues by:

- Keeping records of queue manager changes.
- Keeping records of queue updates for use by the restart process.
- Enabling you to restore data after a hardware or software failure.

This section tells you more about logs, including:

- “What logs look like” on page 126
- “Types of logging” on page 126
- “Checkpointing – ensuring complete recovery” on page 128
- “Media recovery” on page 132
- “Managing logs” on page 130
- “Managing log files” on page 131

What logs look like

An MQSeries log consists of two components:

1. One or more files of log data
2. A log control file

There are a number of log files which contain the data being recorded. You can define the number and size (as explained in Chapter 13, “Configuration files” on page 139), or take the system default of 3 files, each 4MB in size.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used. If you have not changed the log path, they are created in the directory:

```
MQS_ROOT:[MQM.LOG.QmName.ACTIVE]
```

MQSeries starts with these primary log files, but, if the log starts to get full, allocates *secondary* log files. It does this dynamically, and removes them when the demand for log space reduces. By default, up to 2 secondary log files can be allocated, providing a further 8MB of disk space. The default number can also be changed, see Chapter 13, “Configuration files” on page 139.

The log control file contains the information needed to monitor the use of log files: their size and location, the name of the next available file, and so on.

Note: You should ensure that the logs created when you start a queue manager are large enough to accommodate the size and volume of messages that your applications will handle. The default log numbers and sizes will require modification to meet your requirements. How to change the default values is described on page 143.

Types of logging

In MQSeries, the number of files that are used for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities: circular logging 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 128. 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 which are said to be active. Active files contain the log entries required to restart the queue manager. The number of active log files is usually the same as the number of primary log files as defined in the configuration files. (See Chapter 13, “Configuration files” on page 139 for further details of how to define the number.)

The key event that controls whether a log file is termed active or not is a *checkpoint*. An MQSeries checkpoint is a group of log records containing information to allow a successful restart of the queue manager. Any information recorded previously is not required to restart the queue manager and can therefore be termed inactive. (See “Checkpointing – ensuring complete recovery” on page 128 for further information about checkpointing.)

You must decide when inactive log files are no longer required. You may select to archive them, or you may delete them as being no longer of interest to your operation. Refer to “Managing logs” on page 130 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 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.

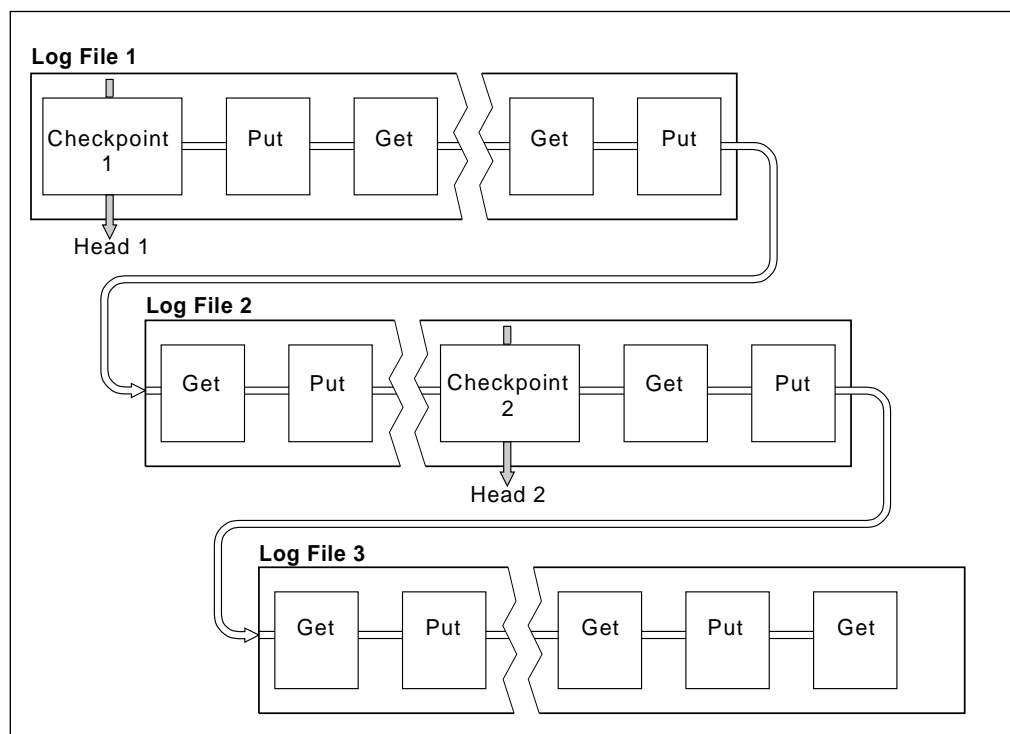


Figure 10. Checkpointing. For simplicity, only the ends of the log files are shown.

In Figure 10, 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.

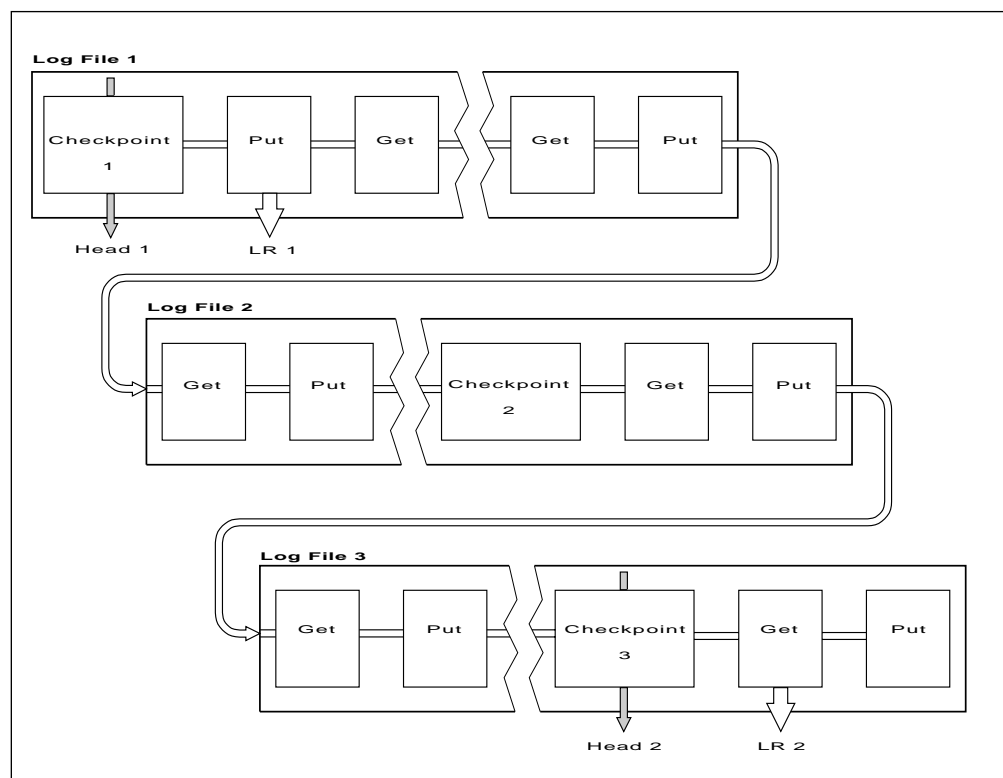


Figure 11. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.

Figure 11 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.

Managing logs

A new primary file is formatted and added to the bottom of the pool in readiness for future logging activities.

Managing logs

Over time, some of the log records written become unnecessary for restarting the queue manager, and the queue manager reclaims freed space in the log files. This activity is transparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log may fill if a checkpoint has not been taken for a long time, or if a long-running transaction wrote a log record a long time ago. The queue manager tries to take checkpoints sufficiently frequently to avoid the first problem.

When a long-running transaction fills the log, attempts to write log records fail and some MQI calls return `MQRC_RESOURCE_PROBLEM`. (Space is reserved to commit or rollback all in-flight transactions, so `MQCMIT` or `MQBACK` should not fail.)

The queue manager rolls back transactions that consume too much log space. An application whose transaction is rolled back in this way is unable to perform subsequent `MQPUT` or `MQGET` operations specifying syncpoint under the same transaction. An attempt to put or get a message under syncpoint in this state returns `MQRC_BACKED_OUT`. The application may then issue `MQCMIT`, which returns `MQRC_BACKED_OUT`, or `MQBACK` and start a new transaction. When the transaction consuming too much log space has been rolled back, its log space is released and the queue manager continues to operate normally.

If the log fills, a message is issued (`AMQ7463`). In addition, if the log fills because a long-running transaction has prevented the space being released, message `AMQ7465` is issued.

Finally, if records are being written to the log faster than the asynchronous housekeeping processes can handle them, message `AMQ7466` is issued. If you see this message, you should increase the number of log files or reduce the amount of data being processed by the queue manager.

What happens when a disk gets full

The queue manager logging component can cope with a full disk, and with full log files. If the disk containing the log fills, the queue manager issues message `AMQ6708` and an error record is taken.

The log files are created at their maximum size, rather than being extended as log records are written to them. This means that MQSeries can only run out of disk space when it is creating a new file. It therefore cannot run out of space when it is writing a record to the log. MQSeries always knows how much space is available in the existing log files and manages the space within the files accordingly.

If you fill the drive containing the log files, you may be able to free some disk space. If you are using a linear log, there may be some inactive log files in the log

directory which you can copy to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You may be able to reduce the number of primary or secondary log files so that the log does not outgrow the available space. Note that it is not possible to alter the size of the log files for an existing queue manager. The queue manager assumes that all log files are the same size.

Managing log files

If you are using circular logging, ensure that there is sufficient space to hold the log files. You do this when you configure your system (see “Log configuration stanzas” on page 144). The amount of disk space used by the log, including the space required for secondary files to be created when required, is limited by the configured size of the disk.

If you are using a linear log, the log files are added continually as data is logged, and the amount of disk space used increases with time. If the rate of data being logged is high, disk space is consumed rapidly by new log files.

Over time, the older log files for a linear log are no longer required to restart the queue manager or perform media recovery of any damaged objects. Periodically, the queue manager issues a pair of messages to indicate which of the log files is required:

- Message AMQ7467 gives the name of the oldest log file needed to restart the queue manager. This log file and all newer log files must be available during queue manager restart.
- Message AMQ7468 gives the name of the oldest log file needed to do media recovery.

Any log files older than these do not need to be online. You can copy them to an archive medium such as tape for disaster recovery, and remove them from the active log directory. Any log files needed for media recovery but not for restart can also be off-loaded to an archive.

If any log file that is needed cannot be found, operator message AMQ6767 is issued. Make the log file, and all subsequent log files, available to the queue manager and retry the operation.

Note: When performing media recovery, all the required log files must be available in the log file directory at the same time. Make sure that you take regular media images of any objects you may wish to recover to avoid running out of disk space to hold all the required log files.

Log file location

When choosing a location for your log files, remember that operation is severely impacted if MQSeries fails to format a new log because of lack of disk space.

If you are using a circular log, ensure that there is sufficient space on the drive for at least the configured primary log files. You should also leave space for at least one secondary log file which is needed if the log has to grow.

If you are using a linear log, you should allow considerably more space; the space consumed by the log increases continuously as data is logged.

Using the log

Ideally, the log files should be placed on a separate disk drive from the queue manager data. This has benefits in terms of performance. It may also be possible to place the log files on multiple disk drives in a mirrored arrangement. This gives protection against failure of the drive containing the log. Without mirroring, you could be forced to go back to the last backup of your MQSeries system.

Using the log for recovery

There are several ways that your data can be damaged. MQSeries for Digital OpenVMS 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 state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when MQSeries stops.

There are ways in which an MQSeries object can become unusable, for example due to inadvertent damage. You then have to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which objects are damaged.

Media recovery

Media recovery means recreating objects from information recorded in a linear log. For example, if an object file is inadvertently deleted, or becomes unusable for some other reason, media recovery can be used to recreate it. The information in the log required for media recovery of an object is called a *media image*. Media images can be recorded manually, using the **rcdmqing** command, or automatically in certain circumstances.

A media image is a sequence of log records containing an image of an object from which the object itself can be recreated.

The first log record required to recreate an object is known as its *media recovery record*; it is the start of the latest media image for the object. The media recovery

record of each object is one of the pieces of information recorded during a checkpoint.

When recreating an object from its media image, it is also necessary to replay any log records describing updates performed on the object since the last image was taken.

Consider, for example, a local queue that has an image of the queue object taken before a persistent message is put onto the queue. In order to recreate the latest image of the object, it is necessary to replay the log entries recording the putting of the message to the queue, as well as replaying the image itself.

When an object is created, the log records written contain enough information to completely recreate the object. These records make up the object's first media image. Subsequently, media images are recorded automatically by the queue manager when:

- Images of all process objects and non-local queues are taken at each shutdown.
- Local queue images are taken when the queue becomes empty.

Media images can also be recorded manually using the **rcdmqimg** command, described in “rcdmqimg (Record media image)” on page 195.

Recovering media images

MQSeries automatically recovers some objects from their media image if it finds that they are corrupt or damaged. In particular, this applies to objects found to be damaged during the normal queue manager startup. If any transaction was incomplete at the time of the last shutdown of the queue manager, any queue affected is also recovered automatically in order to complete the startup operation.

You must recover other objects manually, using the **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 197 for further details of the **rcrmqobj** command.

It is important to remember that you must have the log file containing the media recovery record, and all subsequent log files, available in the log file directory when attempting media recovery of an object. If a required file cannot be found, operator message AMQ6767 is issued and the media recovery operation fails. If you do not take regular media images of the objects that you may wish to recreate, you can get into the situation where you have insufficient disk space to hold all the log files required to recreate an object.

Recovering damaged objects during startup

If the queue manager discovers a damaged object during startup, the action it takes depends on the type of object and whether the queue manager is configured to support media recovery.

If the queue manager object is damaged, the queue manager cannot start unless it can recover the object. If the queue manager is configured with a linear log, and thus supports media recovery, MQSeries automatically tries to recreate the MQSeries object from its media images. If the log method selected does not support media recovery, you can either restore a backup of the queue manager or delete the queue manager.

If any transactions were active when the queue manager stopped, the local queues containing the persistent, uncommitted messages put or got inside these transactions are also needed to start the queue manager successfully. If any of these local queues are found to be damaged, and the queue manager supports media recovery, it automatically attempts to recreate them from their media images. If any of the queues cannot be recovered, MQSeries cannot start.

If any damaged local queues containing uncommitted messages are discovered during startup processing on a queue manager that does not support media recovery, the queues are marked as damaged objects and the uncommitted messages on them are ignored. This is because it is not possible to perform media recovery of damaged objects on such a queue manager and the only action left is to delete them. Message AMQ7472 is issued to report any damage.

Recovering damaged objects at other times

Media recovery of objects is only automatic during startup. At other times, when object damage is detected, operator message AMQ7472 is issued and most operations using the object fail. If the queue manager object is damaged at any time after the queue manager has started, the queue manager 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 197 for further details).

Backup and restore

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 13, “Configuration files” on page 139.

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 42.

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. You can do this with the BACKUP COMMAND and the /BY_OWNER parameter.

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 C, “Directory structure” on page 229 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. Recover the ones you find, using the **rcrmqobj** command, for example:

```
rcrmqobj -m QMgrName -t * *
```

where QMgrName is the queue manager being recovered. -t * * indicates that any object of any type will be recovered. If only one or two objects have been reported as damaged, you may want to specify those objects by name and type here.

Note: These commands do not apply to channels.

- **For linear logging with media recovery and with an undamaged log**, you may be able to restore a backup of the queue manager data leaving the existing log files and log control file unchanged. Starting the queue manager applies the changes from the log to bring the queue manager back to its state when the failure occurred.

This method relies on two facts. Firstly, it is vital that the checkpoint file be restored as part of the queue manager data. This file contains the information determining how much of the data in the log must be applied to give a consistent queue manager.

Secondly, you must have the oldest log file which was required to start the queue manager at the time of the backup, and all subsequent log files, available in the log file directory.

If this is not possible, you must restore a backup of both the queue manager data and the log, both of which were taken at the same time.

- **For circular logging, or linear logging without media recovery**, you must restore the queue manager from the latest backup that you have. Once you have restored the backup, restart the queue manager and check as above for damaged objects. However, because you do not have media recovery, you must find other ways of recreating the damaged objects.

Damaged queue manager object

If the queue manager object has been reported as damaged during normal operation, the queue manager performs a 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. 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.

Recovery scenarios

Chapter 13. Configuration files

MQSeries for Digital OpenVMS uses *configuration files* to hold basic product configuration information. This chapter describes what they are and how you can use them to change the way that queue managers operate. It contains the following sections

- “What configuration files are”
- “MQSeries configuration file”
- “Queue manager configuration file” on page 141
- “Editing configuration files” on page 143
- “Configuring the logs” on page 143
- “Specifying log file sizes” on page 147

What configuration files are

Configuration files define optional values for individual queue managers and for MQSeries on the node as a whole. These files have file name extensions of “ini” and are also referred to as ini files or stanza files.

A configuration file contains one or more *stanzas*, where a stanza is simply a group of lines in the file that together have a common function or define part of a system. For example, there are stanzas associated with logs, with channels, and installable services.

Configuration files may be modified automatically by commands that change the configuration of queue managers on the node and also by editing them manually.

Configuration files can be one of the following:

- The *MQSeries configuration file*, which specifies values for the MQSeries on the node as a whole. There is one MQSeries configuration file per node.
- *Queue manager configuration files*, which specify values for specific queue managers. There is one queue manager configuration file for each queue manager on the node.

MQSeries configuration file

The MQSeries configuration file `mqs.ini` contains information relevant to all the queue managers on a node. It is created automatically during installation. In particular, the MQSeries configuration file is used to locate the data associated with each queue manager. The MQSeries configuration file is located in the `mqm` directory, by default `MQS_ROOT:[MQM]`.

What the MQSeries configuration file contains

The `mqs.ini` file contains the names of the queue managers, the name of the default queue manager, and the location of the files associated with each of them. The following stanzas can appear in `mqs.ini`:

AllQueueManagers

Specifies the path to the `qmgrs` directory where the files associated with a queue manager are stored.

DefaultQueueManager

Specifies the default queue manager for the node. This queue manager processes any commands where a queue manager name is not explicitly specified. The stanza is automatically updated if you create a new default queue manager. If you inadvertently create a default queue manager and then wish to revert to the original, you must alter this stanza manually.

QueueManager

There is one such stanza for each queue manager. This specifies the queue manager name and the location of the files associated with that queue manager. The names of these files are based on the queue manager name but are transformed if the queue manager name is not a valid filename. See “Understanding MQSeries file names” on page 42.

LogDefaults

Specifies the default log parameters for the node. The *DefaultPrefix* and *DefaultPath* entries allow for the queue manager and its log to be on different physical drives. This is recommended, although by default they are on the same drive. See “Configuring the logs” on page 143 for more information about the log file stanzas.

Figure 12 shows an example of an MQSeries configuration file.

```

*****#
#* Module Name: mqs.ini                                     *#
#* Type       : MQSeries Configuration File                *#
#* Function   : Define MQSeries resources for the node    *#
*****#
#* Notes      :                                           *#
#* 1) This is an example MQSeries configuration file      *#
*****#
AllQueueManagers:
*****#
#* The path to the qmgrs directory, below which queue manager data *#
#* is stored                                               *#
*****#
DefaultPrefix=mqs_root:[mqm]

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=mqs_root[mqm.log]

QueueManager:
  Name=saturn.queue.manager
  Prefix=mqs_root:[mqm]
  Directory=saturn!queue!manager

QueueManager:
  Name=pluto.queue.manager
  Prefix=mqs_root:[mqm]
  Directory=pluto$queue$manager

DefaultQueueManager:
  Name=saturn.queue.manager
  
```

Figure 12. Example MQSeries configuration file

In Figure 12, 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.

Attention 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.

Queue manager configuration file

A queue manager configuration file, `qm.ini`, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. It is created automatically when the queue manager with which it is associated is created.

The file is held in the root of the directory tree occupied by the queue manager. For example, the path and name for a configuration file for a queue manager called `QMNAME` is:

```
MQS_ROOT:[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 42.

What the queue manager configuration file contains

The stanzas that can appear in a queue manager configuration file, `qm.ini` are as follows:

Log

Specifies the default log parameters for this queue manager. The fields in this stanza are same as those in the `LogDefaults` stanza in the `mqs.ini` file. The values can be changed, if required. See “Configuring the logs” on page 143 for more information about the log file stanzas.

Channels

This stanza contains information about the channels. As well as defining the maximum number of channels (`MaxChannels`) that can be defined for the queue manager, a second parameter (`MaxActiveChannels`) limits the number of channels that can be active at any time.

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

LU6.2 and TCP

Specifies network protocol configuration parameters. These stanzas override the default parameters for channels. Only stanzas representing changed default values are actually present.

`KeepAlive`, if specified, causes TCP/IP to periodically check that the other end of the connection is still available. If it is not, the channel is closed.

Queue manager configuration file

See the *MQSeries Distributed Queuing Guide* for more information.

Figure 13 shows how the stanzas might be arranged in a queue manager configuration file.

```
#####  
#* Module Name: qm.ini                                     *#  
#* Type       : MQSeries queue manager configuration file *#  
# Function    : Define the configuration of a single queue manager *#  
#*          *#  
#####  
#* Notes      :                                           *#  
#* 1) This file defines the configuration of the queue manager *#  
#*          *#  
#####  
Service:  
  Name=AuthorizationService  
  EntryPoints=9  
  
ServiceComponent:  
  Service=AuthorizationService  
  Name=MQSeries.UNIX.auth.service  
  Module=amqzfu  
  ComponentDataSize=0  
  
Service:  
  Name=NameService  
  EntryPoints=9  
  
ServiceComponent:  
  Service=NameService  
  Name=MQSeries.UNIX.name.service  
  Module=abctest  
  ComponentDataSize=128  
  
Log:  
  LogPrimaryFiles=3  
  LogSecondaryFiles=2  
  LogFilePages=1024  
  LogType=CIRCULAR  
  LogBufferPages=17  
  LogPath=mqs_root:[mqm.log.SATURN$QUEUE$MANAGER]  
  
CHANNELS:  
  MaxChannels = 20           ; Maximum number of Channels allowed,  
                             ; the default number is 100  
  MaxActiveChannels = 10    ; Maximum number of Channels allowed to be  
                             ; active at any time. The default is the  
                             ; value of MaxChannels.  
  
TCP:  
  Port = 1800                ; TCP/IP entries  
  KeepAlive = Yes           ; use port 1800 instead of the default 1414  
                             ; Switch KeepAlive on
```

Figure 13. Example queue manager configuration file

Editing configuration files

You can edit the default configuration files to alter the system defaults. However, before editing any configuration file, make sure that you have a backup that you can revert to.

In some circumstances, you may have to edit your configuration files. For example:

- If you lose a configuration file; recover from backup if possible.
- If you need to move one or more queue managers to a new directory.
- If you need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- When advised to do so by your IBM Support Center.

Changing the default prefix

If you change the default prefix, `DefaultPrefix`, for the message queue manager, you must replicate the directory structure that was created at installation time (see Figure 16 on page 229). In particular, the `qmgrs` structure must be created. You must stop MQSeries before changing the default prefix. Only restart MQSeries after the structures have been moved to the new location and the default prefix has been changed.

Implementing changes to configuration files

If you edit a configuration file, the changes are not implemented immediately by the queue manager. Changes made to the MQSeries configuration file are only implemented when MQSeries is started. Changes made to a queue manager configuration file are implemented when the queue manager is started. If the queue manager is running when you make the changes, you must stop and then restart the queue manager for any changes to be recognized by the system.

Recommendations for configuration files

When you create a new queue manager, you should:

- Back up the MQSeries configuration file
- Back up the new queue manager configuration file

Configuring the logs

The log parameters in the MQSeries configuration file are used as default values when you create a queue manager. These defaults can be overridden if you specify the log parameters on the `crtmqm` command. See “`crtmqm` (Create queue manager)” on page 176 for details of this command.

The values specified in the queue manager configuration file are read when the queue manager is started. The file is created when the queue manager is created.

The values in a configuration file are set according to these priorities:

1. Parameters entered on the command line override both the queue manager configuration file and the MQSeries configuration file.
2. The queue manager configuration file overrides the MQSeries configuration file.

Configuring logs

3. The MQSeries configuration file contains the supplied default values.

Note: Resource “MQM” must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the logs files are in the default locations supplied with the product.

If you use a value that is not valid in a configuration file, it is ignored. The effect is the same as missing out the value entirely. An operator message is issued to indicate the problem.

You can edit the MQSeries configuration file after installation and change the default values to your own requirements.

Log configuration stanzas

The size and location of the log is configured by stanzas in the MQSeries and queue manager configuration files. These stanzas specify the type of logging to be used, the log file size, and the log path.

The MQSeries configuration file contains a stanza called *LogDefaults* with the following format:

```
LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=17
  LogDefaultPath=MQS_ROOT:[MQM.LOG]
```

The values specified in the MQSeries configuration file are read whenever a queue manager is created, started, or deleted.

Each queue manager configuration file has a stanza called *Log*, which has the following format:

```
Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=17
  LogPath=MQS_ROOT:[MQM.LOG.<QM_Dir_Name>]
```

<QM_Dir_Name> is the subdirectory name for this queue manager, providing a unique path to the logs. This is the queue manager name if it is valid for the file system; otherwise, it is a transformed name. See “Understanding MQSeries file names” on page 42.

LogPrimaryFiles

Primary log files are the log files allocated during creation for future use.

The default number is 3. The default can be overridden by editing the *LogPrimaryFiles* value in the product and queue manager configuration files.

The value is examined when the queue manager is created or started. You can increase or decrease it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted and the effect may not be immediate.

The minimum number of primary log files is 2 and the maximum is 62. The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogSecondaryFiles

Secondary log files are the log files allocated when the primary files are exhausted.

The default number is 2. The default can be overridden using the *LogSecondaryFiles* value in the product and queue manager configuration files.

The value is examined when the queue manager is created or started. You can change this value, but changes are not effective until the queue manager is restarted, and the effect may not be immediate.

The minimum number of secondary log files is 1 and the maximum is 61. The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

LogFilePages

The log data is held in a series of files called log files.

The default number of log file pages is 1024, equating to a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 16 384.

The log file size is specified in units of 4 KB pages. It can be specified only during queue manager creation and the value used is obtained by taking the default (1024) and overriding it with the value in the *LogFilePages* attribute in the MQSeries configuration file, or by overriding with the value specified on the **crtmqm** command using the **-lf** flag.

Note: The size of the log files is specified during queue manager creation and cannot be changed for an existing queue manager.

LogType

The *LogType* parameter is used to define the type to be used, either CIRCULAR or LINEAR. The default is CIRCULAR.

If you want to change the default, you can either edit the MQSeries configuration file or specify linear logging with the **crtmqm** command. You cannot change the logging method after a queue manager has been created.

LogBufferPages

The amount of memory allocated to buffer records for writing is configurable. The size of the buffers is specified in units of 4 KB pages.

The default number of buffer pages is 17, equating to 68 KB.

The default can be overridden using the *LogBufferPages* value in the MQSeries and queue manager configuration files.

Configuring logs

The value is examined when the queue manager is created or started and may be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

The minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

LogPath

You can specify the directory in which the log files for a queue manager reside. The directory should exist on a local device to which the queue manager can write and, preferably, should be on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is `MQS_ROOT:[MQM.LOG]`

You can specify the name of a directory in the **crtmqm** command using the `-ld` flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the Log File Path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify `-ld` on the **crtmqm** command, the value of the *LogDefaultPath* attribute in the MQSeries configuration file is used. If this attribute is missing, the default of `MQS_ROOT:[MQM.LOG]` is used. The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager has been created, a *LogPath* value is created in the log stanza in the queue manager configuration file giving the complete directory name for the queue manager's log. This value is used to locate the log when the queue manager is started or deleted.

Specifying log file sizes

The size of the log file that you require depends on the number and size of messages that are to be handled by your system. Each operation adds an overhead to the size of the log. For example, when a persistent message is put to a queue, the message data must be written to the log to make recovery of the message possible. The message descriptor is also logged together with some internal information that describes the effect of putting the message on the queue.

There is a trade-off between the size of your log files and the number of files that you have. Larger files are more difficult to handle but are more efficient.

Table 10 shows approximate values for the header information required for various types of operation.

Operation	Size
Put persistent message	600 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, roll-back	1000 bytes + 12 bytes for each get or put to be rolled back
Create object	1500 bytes
Delete object	300 bytes
Alter attributes	1024 bytes
Record media image	800 bytes + image The image is divided into segments of 15700 bytes, each having a 300-byte overhead.
Checkpoint	750 bytes + 200 bytes for each active unit of work. Additional data may be logged for any uncommitted puts or gets that have been buffered for performance reasons.

Specifying log file sizes

Chapter 14. Problem determination

This chapter suggests reasons for some of the problems you may have using MQSeries for Digital OpenVMS. 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”
- “Common programming errors” on page 152
- “What to do next” on page 153
- “Application design considerations” on page 156
- “Incorrect output” on page 157
- “Error logs” on page 160
- “Dead-letter queues” on page 163
- “Configuration files and problem determination” on page 164
- “Using MQSeries trace” on page 164
- “First failure support technology (FFST)” on page 165
- “Problem determination with clients” on page 166

Preliminary checks

The cause of your problem could be in:

- MQSeries
- The network
- The application

The sections that follow raise some fundamental questions that you need to consider. Work through the questions, making a note of anything that might be relevant to the problem.

Has MQSeries run successfully before?

If MQSeries has not run successfully before, it is likely that you have not yet set it up correctly. See Chapter 2, “Installing MQSeries for Digital OpenVMS” on page 13 to check that you have carried out all the steps correctly.

Are there any error messages?

MQSeries uses error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use. Check the error logs to see if any messages have been recorded that are associated with your problem.

See “Error logs” on page 160 for information about the contents of the error logs, and their locations.

Are there any return codes explaining the problem?

If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, refer to the *MQSeries Application Programming Reference* manual for a description of that return code.

Can you reproduce the problem?

If you 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?

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 linker, if applicable, to see if any errors have been reported.

If your application fails to translate, compile, or link, it will also fail to run if you attempt to invoke it. See the *MQSeries Application Programming Reference* manual for information about building your application.

If the documentation shows that each of these steps was accomplished without error, you should consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Common programming errors” on page 152 for some examples of common errors that cause problems with MQSeries applications.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of MQSeries has been started.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes, 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 patch error.
- If a patch has been applied to any other program, consider the effect it might have on the way MQSeries interfaces with it.

Common programming errors

The 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.

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 xii) 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?”
- “Have you failed to receive a response from a PCF command?”
- “Does the problem affect only remote queues?” on page 155
- “Is your application or MQSeries for Digital OpenVMS running slowly?” on page 156

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 160 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.

What next

- If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?

The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure (MQDLH).

If the dead-letter queue contains messages, you can use the provided browse sample application (amqsbcg) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.
- Has a message been sent to the error log?

See “Error logs” on page 160 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?

If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See the *MQSeries Application Programming Reference* manual for information about the *WaitInterval* field, and completion and reason codes from MQGET.)
- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to take a syncpoint?

Unless you have specifically excluded your request message from syncpoint, you need to take a syncpoint before attempting to receive reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the *CorrelId* and *MsgId* fields correctly?

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 and are triggerable, 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 channel initiator is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually. See the *MQSeries Distributed Queuing Guide* for information about how to do this.

Application design considerations

See the *MQSeries Distributed Queuing Guide* for information about how to define channels.

Is your application or MQSeries for Digital OpenVMS 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.

A performance problem may be caused by a limitation of your hardware.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that MQSeries is running slowly:

- Your system is slow to respond to MQSeries commands.
- Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem may lie with MQSeries for Digital OpenVMS 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 the *MQSeries Application Programming Guide*.

Effect of message length

Although MQSeries allows messages to hold up to 4MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, you should send only the essential data in a message; for example, in a request to debit a bank account, the only information that may need to be passed from the client to the server application is the account number and the amount of the debit.

Effect of message persistence

Persistent messages are logged. Logging messages reduces the performance of your application, so you should use persistent messages for essential data only. If the data in a message can be discarded if the queue manager stops or fails, use a nonpersistent message.

Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (*MsgId* and *CorrelId*) in the message descriptor to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the *BufferLength* field set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the *MaxMsgLength* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be as high as 4MB, the maximum allowed by MQSeries for Digital OpenVMS.

Frequency of syncpoints

Programs that issue numerous MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently inaccessible, while other tasks might be waiting to get these messages. This has implications in terms of storage, and in terms of threads tied up with tasks that are attempting to get messages.

Use of the MQPUT1 call

Use the MQPUT1 call only if you have a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

Incorrect output

The term “incorrect output” can be interpreted in many different ways. For the purpose of problem determination within this book, the meaning is explained in “Have you obtained incorrect output?” on page 153.

Incorrect output

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.
- 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 multi-server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If this is the case, refer to “Messages that contain unexpected or corrupted information.”

Messages that contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following 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, and the channels between the two queue managers, are active.

- Is triggering set on in the sending system?
- Is the message you are waiting for a reply message from a remote system?
Check that triggering is activated in the remote system.
- Is the queue already full?

This could mean that an application was unable to put the required message onto the queue. If this is so, check if the message has been put onto the dead-letter queue.

The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure.

- Is there a mismatch between the sending and receiving queue managers?
For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?
For example, a mismatch in sequence number wrap stops the distributed queuing component. See the *MQSeries Distributed Queuing Guide* for more information about distributed queuing.
- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic conversion occurs when the MQGET is issued if the format is recognized as one of the built-in formats.

If the data set is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

An exception to the above occurs if you are sending data to MQSeries for MVS/ESA.

Refer to the *MQSeries Distributed Queuing 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.

- If the queue manager name is known and the queue manager is available:
MQS_ROOT: [MQM.QMGRS.QMGrName.ERRORS]AMQERR01.LOG
- If the queue manager is not available:
MQS_ROOT: [MQM.QMGRS.\$SYSTEM.ERRORS]AMQERR01.LOG
- If an error has occurred with a client application:
AMQERR01.LOG
in MQS_ROOT: [MQM.ERRORS]
- First Failure Support Technology (FFST) – see “How to examine the FFSTs” on page 165.

Log files

At installation time an [MQM.QMGRS.\$SYSTEM.ERRORS] directory is created in the QMGRS file path. The errors subdirectory can contain up to three error log files named:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files have the same names as the \$SYSTEM ones, that is AMQERR01, AMQERR02, and AMQERR03, and each has a capacity of 256KB. The files are placed in the errors subdirectory of each queue manager that you create.

As error messages are generated they are placed in AMQERR01. When AMQERR01 gets bigger than 256KB it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager's errors files unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the [MQM.QMGRS.\$SYSTEM.ERRORS] subdirectory.

To examine the contents of any error log file, use your usual OpenVMS editor.

Batch-job log files

Commands that initiate batch jobs (**runmqchl**, **runmqchi**, **runmqdlq**, and **runmqlsr**) create error logs in the MQS_ROOT: [MQM.ERRORS] directory.

Error logs

Early errors

There are a number of special cases where the above error logs have not yet been established and an error occurs. MQSeries attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, due to a corrupt configuration file for example, no location information can be determined, errors are logged to an errors directory that is created at installation time on the root directory, mqm.

If the MQSeries configuration file is readable, and the DefaultPrefix attribute of the AllQueueManagers stanza is readable, errors are logged in the DefaultPrefix[.errors] directory.

For further information about configuration files, see Chapter 13, "Configuration files" on page 139.

Operator messages

In MQSeries for Digital OpenVMS, operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national language (NLS) enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any, and are also written to the error log AMQERR01.LOG in the queue manager directory. For example:

```
MQS_ROOT: [MQM.QMGRS.QUEUE$MANAGER.ERRORS]
```

Some errors are logged to the AMQERR01.LOG file in the queue manager directory and others to the \$SYSTEM directory copy of the error log.

Example error log

This example shows part of an MQSeries for Digital OpenVMS error log:

```

...
08/01/95 11:41:56 AMQ8003: MQSeries queue manager started.
EXPLANATION: MQSeries queue manager janet started.
ACTION: None.
-----
08/01/95 11:56:52 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-----
08/01/95 11:57:26 AMQ9208: Error on receive from host 'camelot
(9.20.12.34)'.
EXPLANATION: An error occurred receiving data from 'camelot
(9.20.12.34)' over TCP/IP. This may be due to a communications failure.
ACTION: Record the TCP/IP return code 232 (X'E8') and tell the
systems administrator.
-----
08/01/95 11:57:27 AMQ9999: Channel program ended abnormally.
EXPLANATION: Channel program 'JANET' ended abnormally.
ACTION: Look at previous error messages for channel program
'JANET' in the error files to determine the cause of the failure.
-----
08/01/95 14:28:57 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
-----
08/02/95 15:02:49 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-----
08/02/95 15:02:51 AMQ9001: Channel program ended normally.
EXPLANATION: Channel program 'JANET' ended normally.
ACTION: None.
-----
08/02/95 15:09:27 AMQ7030: Request to quiesce the queue manager
accepted. The queue manager will stop when there is no further
work for it to perform.
EXPLANATION: You have requested that the queue manager end when
there is no more work for it. In the meantime, it will refuse
new applications that attempt to start, although it allows those
already running to complete their work.
ACTION: None.
-----
08/02/95 15:09:32 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
...

```

Dead-letter queues

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the 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.

Using MQSeries trace

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 have a dead-letter queue on each queue manager you are using. The supplied sample program `amqscoma.tst` creates the default objects for a queue manager, including a dead-letter queue called `SYSTEM.DEAD.LETTER.QUEUE`.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found and result in “queue manager unavailable” type errors.

There are several checks you can make on the configuration files:

- Ensure that the configuration files exist.
- Ensure that they have appropriate permissions, for example:

```
MQS.INI;1 MQM (RWED, RWED, RWED, RE )
          (identifier=MQM, access=READ+WRITE+EXECUTE+DELETE+CONTROL) .
```
- Ensure that the MQSeries configuration file references the correct queue manager and log directories.

Using MQSeries trace

MQSeries for Digital OpenVMS uses the following commands for the trace facility:

- **strmqtrc** – see “strmqtrc (Start MQSeries trace)” on page 219
- **dspmqrtrc** – see “dspmqrtrc (Display MQSeries formatted trace output)” on page 189
- **endmqtrc** – see “endmqtrc (End MQSeries trace)” on page 194

The trace facility uses one file for each entity being traced, with the trace information being recorded in the appropriate file.

Files associated with trace are created in the directory `MQS_ROOT:[MQM.TRACE]`.

The files in this directory include details of queue managers, as well as all early tracing and all \$SYSTEM tracing.

Trace file names

Trace file names are constructed in the following way:

`AMQppppp.TRC`

where `ppppp` is the process identifier (PID) of the process producing the trace.

Notes:

1. The value of the process identifier can contain fewer, or more, digits than shown in the example.
2. There will be one trace file for each process running as part of the entity being traced.

Sample trace data

The following sample is an extract from a Digital OpenVMS trace:

ID	ELAPSED_MSEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
30d	0	0	MQS CEI Exit!.	12484.1	xcsWaitEventSem	rc=10806020
30d	0	0	MQS CEI Exit!	12484.1	zcpReceiveOnLink	rc=20805311
30d	0	0	MQS FNC Entry	12484.1	zxcProcessChildren	
30d	0	0	MQS CEI Entry.	12484.1	xcsRequestMutexSem	
30d	1	0	MQS CEI Entry..	12484.1	xcsHSHMEMBtoPTR	
30d	1	0	MQS CEI Exit...	12484.1	xcsHSHMEMBtoPTR	rc=00000000
30d	1	0	MQS FNC Entry..	12484.1	x11SemGetVal	
30d	1	0	MQS FNC Exit...	12484.1	x11SemGetVal	rc=00000000
30d	1	0	MQS FNC Entry..	12484.1	x11SemReq	
30d	1	0	MQS FNC Exit...	12484.1	x11SemReq	rc=00000000
30d	1	0	MQS CEI Exit..	12484.1	xcsRequestMutexSem	rc=00000000
30d	2	0	MQS CEI Entry.	12484.1	xcsReleaseMutexSem	
30d	2	0	MQS CEI Entry..	12484.1	xcsHSHMEMBtoPTR	
30d	2	0	MQS CEI Exit...	12484.1	xcsHSHMEMBtoPTR	rc=00000000
30d	2	0	MQS FNC Entry..	12484.1	x11SemRel	
30d	2	0	MQS FNC Exit...	12484.1	x11SemRel	rc=00000000
30d	2	0	MQS CEI Exit..	12484.1	xcsReleaseMutexSem	rc=00000000
30d	2	0	MQS CEI Entry.	12484.1	xcsHSHMEMBtoPTR	
...						

Figure 14. Sample MQSeries for Digital OpenVMS 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)

Information which, on the OS/2 and AIX platforms, is normally recorded in FFST logs is, on MQSeries for Digital OpenVMS, recorded in a file in the MQS_ROOT:[MQM.ERRORS] directory.

These errors are normally severe, unrecoverable errors and indicate either a configuration problem with the system or an MQSeries internal error.

How to examine the FFSTs

The files are named AMQnnnnn_mm.FDC, where:

nnnnn Is the process id reporting the error
mm Is a sequence number, normally 0

When a process creates an FFST it also writes an entry in the system error log. The record contains the name of the FFST file to assist in automatic problem tracking.

A typical FFST is shown in Figure 15.

Client problem determination

```
MQSeries First Failure Symptom Report
=====
Date/Time      :- Friday July 14 14:06:52 BST 1995
Host Name      :- unknown
PIDS           :- 5697175
LVLS          :- 220
Product Long Name :- MQSeries for Digital OpenVMS
Vendor         :- IBM
Probe Id       :- XC130003
Application Name :- MQM
Component      :- xehExcepti
Build Date     :- Jul 14 1995
Userid        :- 00000231 (mqm)
Process       :- 00015967
Major Errorcode :- xecSTOP
Minor Errorcode :- OK
Probe Type     :- HALT6109
Probe Severity :- 1
Probe Description :- AMQ6125: An internal MQSeries error has occurred.
Arith1        :- 11 b
```

```
MQM Function Stack
x11TidyUpSems
xcsFFST
```

```
MQM Trace History
...
```

Figure 15. Sample MQSeries for Digital OpenVMS 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 is generated, apart from raising problems through the support centers.

However, there is one set of problems that they may be able to solve. If the FFST shows “quota exceeded” or “out of space on device” descriptions when calling one of the internal functions, it is likely that the relevant SYSGEN parameter limit has been exceeded.

To resolve the problem, adjust the system parameters to increase the internal limits. See “System configuration” on page 21 for further details.

Problem determination with clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are now additional reason codes for error conditions associated with clients. For example:

- Remote machine not responding
- Communications line error
- Invalid machine address

The most common time for errors to occur is when an application issues an MQCONN and receives the response MQRC_Q_MQR_NOT_AVAILABLE. An error message, written to the client log file, explains the cause of the error. Messages may also be logged at the server depending on the nature of the failure.

Terminating clients

Even though a client has terminated it is still possible for the process at the server to be holding its queues open. Normally, this will only be for a short time until the communications layer detects that the partner has gone.

Error messages with clients

When an error occurs with a client system, error messages are put into the error files associated with the server, if possible. If 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, UNIX and OpenVMS systems clients

Error messages for OS/2, and UNIX systems, clients are placed in the error logs on their respective MQSeries server systems. Typically, these files appear in the MQS_ROOT:[MQM.ERRORS] directory.

DOS and Windows clients

The location of the log file AMQERR01.LOG is set by the MQDATA environment variable. The default location, if not overridden by MQDATA, is:

C:\

Working in the DOS environment involves the environment variable MQDATA.

This is the default library used by the client code to store trace and error information; it also holds the directory name in which the qm.ini file is stored. (needed for NetBIOS setup). If not specified, it defaults to the C drive.

The names of the default files held in this library are:

- AMQERR01.LOG** For error messages.
- AMQERR01.FDC** For First Failure Data Capture messages.

Client problem determination

Part 2. Reference

Chapter 15. MQSeries control commands

This chapter contains reference material for the control commands used with MQSeries for Digital OpenVMS. All commands in this chapter can be issued from an OpenVMS DCL prompt. These commands are case-sensitive.

Names

In general, the names of MQSeries objects can have up to 48 characters. This rule applies to all the following objects:

- Queue managers
- Queues
- Process definitions

The maximum length of channel names is 20 characters.

The characters that can be used for all MQSeries names are:

- Uppercase A–Z
- Lowercase a–z
- Numerics 0–9
- Period (.)
- Underscore (_)
- Forward slash (/) (see note 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 only essential if special characters are included in the name, or if case needs to be preserved.

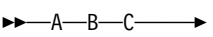
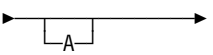
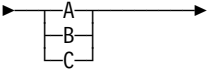
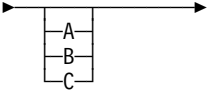
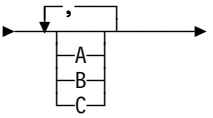
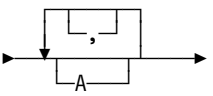
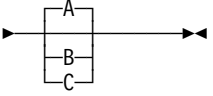
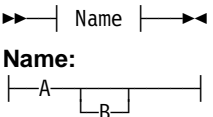
How to read syntax diagrams

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 11. 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.
	You may specify value A multiple times. The separator in this example is optional.
	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. Name: —A— —B—
Punctuation and uppercase values	Specify exactly as shown.
Lowercase values (for example, <i>name</i>)	Supply your own text in place of the <i>name</i> variable.

Syntax help

You can obtain help about the syntax of any of the commands in this chapter by entering the command followed by a question mark. MQSeries responds by listing the syntax required for the selected command. The syntax shows all the parameters and variables associated with the command. Different forms of parentheses are used to indicate whether a parameter is required or not. For example:

```
CmdName [-x OptParam ] ( -c | -b ) { -p principal } argument
```

where:

CmdName Is the command name for which help has been requested.

[-x OptParam] The square brackets indicate that this is an optional parameter.

(-c | -b) A mandatory field. In this case, you must select one of the flags c or b.

{ -p principal }
 An optional list of variables that you may supply, but, if this is shown, at least one variable must be provided when you enter the command.

argument An argument required to be supplied with this command, mandatory if shown on the response to the query.

Examples

1. Result of entering `endmqm` ?

```
endmqm [-z] [-c | -i | -p] QMgrName
```

2. Result of entering `rcdmqimg` ?

```
rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]
```

MQSeries return codes

Most of the MQSeries commands, for example `crtmqm`, write a status line when ending to indicate the success or failure of the command.

If the status of a command is to be tested in a DCL command file, it may be necessary to interpret the status value returned from an MQSeries program.

The MQSeries return codes are defined in a message file called `SYS$MESSAGE:MQS_MSG.EXE`.

In order to access the message text associated with a return code in the file, you **must** use the DCL SET MESSAGE command. This command loads the message codes into the message table of your process. For example:

```
$ SET MESSAGE SYS$MESSAGE:MQS_MSG.EXE
```

After this, you can use the `F$MESSAGE` lexical function to print the text of an MQSeries return code. For example:

```
$ strmqm )(*bad-qm-name&##
The queue manager name is either not valid or not known
$ WRITE SYS$OUTPUT F$MESSAGE($STATUS)
%MQS-F-CSPRC_Q_MGR_NAM, Queue manager name error
```

In order to convert the OpenVMS return code to a return code value used in MQSeries for OS/2 or UNIX systems, you can use the following DCL equation:

```
$ RC = $STATUS / 8 .AND. %xFFF
```

For example:

```
$ crtmqm &*)*(
The queue manager name is either not valid or not known
$ RC = $STATUS / 8 .AND. %xFFF
$ SHOW SYMBOL RC
RC = 72    Hex = 00000048    Octal = 0000000110
```

crtmqcvx (Data conversion)

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert your C structures.

The command reads an input file containing a structure or structures to be converted. It then writes an output file containing a code fragment or fragments to convert those structures.

For further information about this command and how to use it, refer to the *MQSeries Application Programming Guide*.

Syntax

```
▶▶—crtmqcvx—SourceFile—TargetFile————▶▶
```

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, you should understand that the fragment uses macros supplied in the MQSeries header file amqsvmha.h.

```

MQLONG Convertmy_structure(
    PMQBYTE *in_cursor,
    PMQBYTE *out_cursor,
    PMQBYTE in_lastbyte,
    PMQBYTE out_lastbyte,
    MQHCONN hConn,
    MQLONG  opts,
    MQLONG  MsgEncoding,
    MQLONG  ReqEncoding,
    MQLONG  MsgCCSID,
    MQLONG  ReqCCSID,
    MQLONG  CompCode,
    MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}

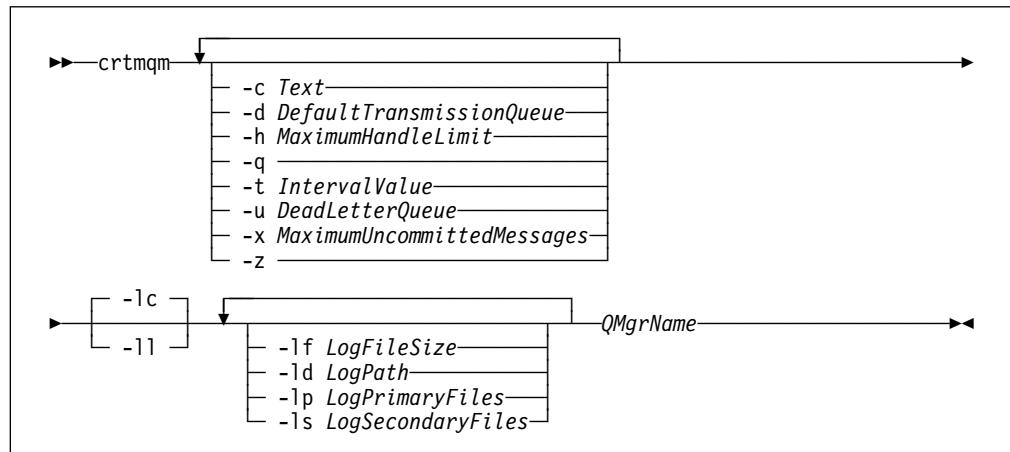
```

crtmqm (Create queue manager)

Purpose

Use the **crtmqm** command to create a local queue manager. Once a queue manager has been created, use the **strmqm** command to start it.

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*

Specify some descriptive text for this queue manager. The default is all blanks.

You can use up to 64 characters. If mixed case is required, the description must be enclosed in double quotes.

-d *DefaultTransmissionQueue*

Specifies the name of the local transmission queue that remote messages are placed on if a transmission queue is not explicitly defined for their destination. There is no default.

-h *MaximumHandleLimit*

Specifies the maximum number of handles that any one application can have open at the same time.

Specify a value in the range 1 through 999 999 999. The default value is 256.

-q Specifies that this queue manager is to be made the default queue manager. The new queue manager replaces any existing queue manager as the default.

If you accidentally use this flag and wish to revert to an existing queue manager as the default queue manager, you can edit the *DefaultQueueManager* stanza in the MQSeries configuration file. See

Chapter 13, “Configuration files” on page 139 for information about configuration files.

-t *IntervalValue*

Specifies the trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after the receipt of a trigger generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You may wish to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be taken effectively means that triggering is disabled after the first trigger message. However, triggering can be reenabled by an application servicing the queue using an alter queue command to reset the trigger attribute.

-u *DeadLetterQueue*

Specifies the name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default if the attribute is omitted is no dead-letter queue.

-x *MaximumUncommittedMessages*

Specifies the maximum number of uncommitted messages under any one syncpoint. That is, the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside a syncpoint.

Specify a value in the range 1 through 10 000. The default value is 1000 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.

The following set of flags is used to define the logging to be used by the queue manager being created. For more information about logs, see “Using the log for recovery” on page 132.

-lc Circular logging is to be used. This is the default logging method.

-ll Linear logging is to be used.

-lf *LogFileSize*

Specifies the size of the log files in units of 4 KB. The minimum value is 64, and the maximum is 16384. The default value is 1024, giving a default log size of 4 MB.

-ld *LogPath*

Specifies the directory to be used to hold the log files. The default is [MQM.LOG]. The default can also be changed when MQSeries is customized.

User ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is done automatically if the log files are in their default locations.

-lp *LogPrimaryFiles*

Specifies the number of primary log files to be allocated. The default value is 3, the minimum is 2, and the maximum is 62.

-ls *LogSecondaryFiles*

Specifies the number of secondary log files to be allocated. The default value is 2, the minimum is 1, and the maximum is 61.

Note: The total number of log files is restricted to 63, regardless of the number requested.

Return codes

- 0** Queue manager created
- 8** Queue manager already exists
- 49** Queue manager stopping
- 69** Storage not available
- 70** Queue space not available
- 71** Unexpected error
- 72** Queue manager name error
- 100** Log location invalid
- 111** Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect.
- 115** Invalid log size

Examples

1. This command creates a default queue manager named `Paint.queue.manager`, which is given a description of `Paint shop`. It also specifies that linear logging is to be used:

```
crtmqm -c "Paint shop" -ll -q "Paint.queue.manager"
```

2. This example requests a number of log files. Two primary and three secondary log files are specified.

```
crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q "Paint.queue.manager"
```

3. In this example, another queue manager, `travel`, is created. The trigger interval is defined as 5000 milliseconds (or 5 seconds) and its dead-letter queue is specified as `SYSTEM.DEAD.LETTER.QUEUE`.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
```

Once a trigger event has been generated, further trigger events are disabled for five seconds.

Related commands

strmqm	Start queue manager
endmqm	End queue manager
dltmqm	Delete queue manager

dltmqm (Delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager. All objects associated with this queue manager are also deleted. Before you can delete a queue manager you must end it using the **endmqm** command.

Syntax

```
▶▶ dltmqm [-z] QMgrName ▶▶
```

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

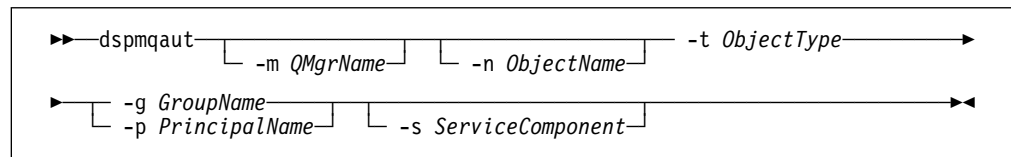
dspmqaout (Display authority)

Purpose

Use the **dspmqaout** command to display the current authorizations to a specified object. Only one group may be specified.

If a user ID is a member of more than one group, examine the authorizations of each group to determine all the authorizations that apply to the user ID.

Syntax



Required parameters

- t *ObjectType*
Specifies the type of object on which the inquiry is to be made. Possible values are:
 - queue** or **q** A queue or queues matching the object type parameter
 - qmgr** A queue manager object
 - process** or **prcs** A process

Optional parameters

- m *QMgrName*
Specifies the name of the queue manager on which the inquiry is to be made.
- n *ObjectName*
Specifies the name of the object on which the inquiry is to be made.
This is a required parameter **unless** it is the queue manager itself.
You must specify the name of a queue manager, queue, or process definition.
- g *GroupName*
Specifies the name of the user group on which the inquiry is to be made.
You can only specify **one** name, which must be the name of an existing rights identifier
- p *PrincipalName*
Specifies the name of a user whose authorizations to the specified object are to be displayed.
- s *ServiceComponent*
This parameter only applies if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

Returned parameters

This command returns an authorization list, which can contain none, one, or more authorization parameters. Each authorization parameter returned means that any user ID in the specified group has the authority to perform the operation defined by that parameter.

Table 12 shows the authorities that can be given to the different object types.

Authority	Queue	Process	Qmgr
all	√	√	√
alladm	√	√	√
allmqi	√	√	√
altusr			√
browse	√		
chg	√	√	√
clr	√		
connect			√
cpy	√	√	√
crt	√	√	√
dlt	√	√	√
dsp	√	√	√
put	√		
inq	√	√	√
get	√		
passall	√		
passid	√		
set	√	√	√
setall	√		√
setid	√		√

The following list defines the authorizations associated with each parameter:

- all** Use all operations relevant to the object.
- alladm** Perform all administration operations relevant to the object.
- allmqi** Use all MQI calls relevant to the object.
- altusr** Specify an alternate user ID on an MQI call.
- browse** Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
- chg** Change the attributes of the specified object, using the appropriate command set.

chgaut	Specify authorizations for other groups of users on the object, using the setmqaut command.
clr	Clear a queue (PCF command Clear queue only).
connect	Connect the application to the specified queue manager by issuing an MQCONN call.
cpy	Copy the definition of an object, for example, the PCF Copy queue command.
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:

```
dspmqaout -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

```
▶▶—dspmqcsv—QMgrName—————▶▶
```

Required 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

dspmqls (Display MQSeries files)

Purpose

Use the **dspmqls** 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 42 for further information about name transformation.

Syntax

```

  ► dspmqls [-m QMgrName] [-t ObjType] GenericObjName ►

```

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

Note: The **dspmqls** command displays the directory containing the queue, *not* the name of the queue itself.

dspmqfls

Return codes

- 0** Command completed normally
- 10** Command completed but not entirely as expected
- 20** An error occurred during processing

Examples

1. The following command displays the details of all objects with names beginning SYSTEM.ADMIN that are defined on the default queue manager.

```
dspmqfls SYSTEM.ADMIN*
```

2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

```
dspmqfls -m RADIUS -t prcs PROC*
```

dspmqtrc (Display MQSeries formatted trace output)

Purpose

Use the **dspmqtrc** command to display MQSeries formatted trace output.

Syntax

```

▶▶—dspmqtrc—┌—t FormatTemplate—┐—InputFileName—◀◀

```

Required parameters

InputFileName

Specifies the name of the file containing the unformatted trace. For example
MQS_ROOT:[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 SYS\$SHARE:AMQTRC.FMT.

Related commands

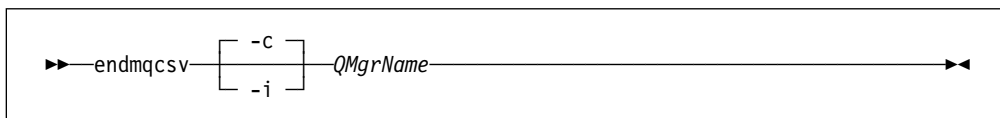
endmqtrc End MQSeries trace
strmqtrc Start MQSeries trace

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 |

endmqm (End queue manager)

Purpose

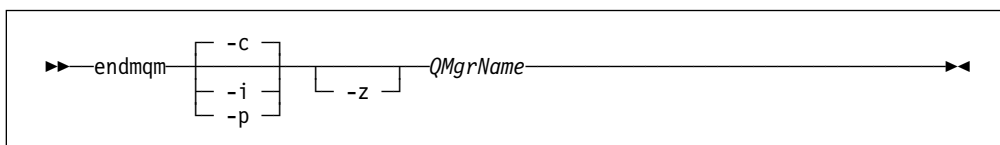
Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Normal or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, you must stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Required parameters

QMgrName

Specifies the name of the message queue manager to be stopped.

Optional parameters

- c** Controlled (or quiesced) shutdown. The queue manager stops but only after all applications have disconnected. Any MQI calls currently being processed are completed. This is the default.
- i** Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.
- p** Preemptive shutdown.
Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.
The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for 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

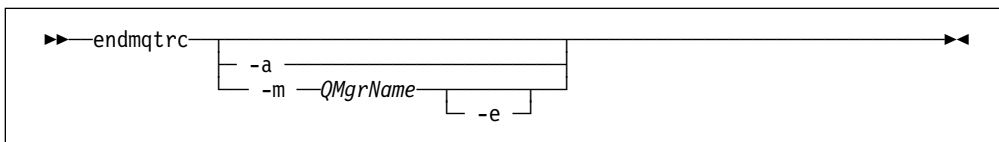
crtmqm	Create a queue manager
strmqm	Start a queue manager
dltmqm	Delete a queue manager

endmqtrc (End MQSeries trace)

Purpose

Use the **endmqtrc** command to end tracing for the specified entity or all entities.

Syntax



Optional parameters

-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

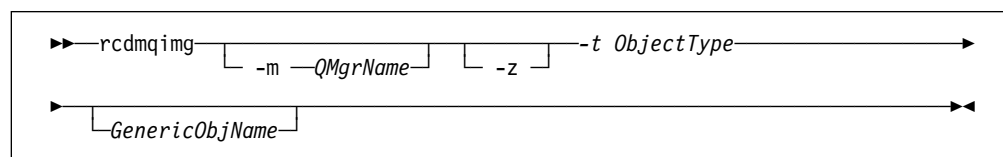
rcdmqimg (Record media image)

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

-t *ObjectType*

Specifies the type of objects whose images are to be recorded. Valid object types are:

prcs or process	Processes
q or queue	All types of queue
ql or qlocal	Local queues
qa or qalias	Alias queues
qr or qremote	Remote queues
qm or qmodel	Model queues
qmgr	Queue manager object
* or all	All of the above

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.

GenericObjName

Specifies the name of the object that is to be recorded. This parameter may have a trailing asterisk to indicate that any objects with names matching the portion of the name prior to the asterisk are to be recorded.

This parameter is required **unless** you are recording a queue manager.

rcdmqimg

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

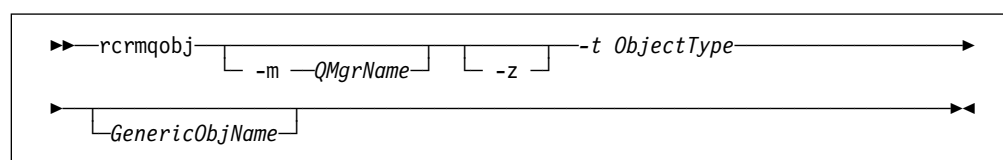
rcrmqobj (Recreate object)

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

-t ObjectType

Specifies the type of objects to be recreated. Valid object types are:

prcs or process	Processes
q or queue	All types of queue
ql or qlocal	Local queues
qa or qalias	Alias queues
qr or qremote	Remote queues
qm or qmodel	Model queues
* or all	All the above
syncfile	The channel synchronization file

Note: Using this flag causes the channel synchronization file to be regenerated for the queue manager specified. This is necessary because the file is not saved by the **rcdmqimg** command.

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.

GenericObjName

Specifies the name of the object that is to be recreated. This parameter may have a trailing asterisk to indicate that any objects with names matching the portion of the name prior to the asterisk are to be recreated.

rcrmqobj

This parameter is required *unless* the object type is the channel synchronization file. If an object name is supplied for this type, it is ignored.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
66	Media image not available
68	Media recovery is not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
135	Temporary object cannot be recovered
136	Object in use

Examples

1. The following command recreates all local queues for the default queue manager:

```
rcrmqobj -t ql *
```

2. The following command recreates all remote queues associated with queue manager store:

```
rcrmqobj -m store -t qr
```

Related commands

rcdmqimg Record an MQSeries object in the log

rsvmqtrn (Resolve MQSeries transactions)

Purpose

Use the **rsvmqtrn** command to give a commit or backout decision to an in-doubt transaction.

Notes:

1. This command must be used only in situations where you are certain that the transaction will not be resolved by the normal protocols. Issuing this command may result in the loss of transactional integrity between resource managers for a distributed transaction.
2. The only time that you can expect to use this command is if you are using an external transaction manager and are involved with two-phase commitment procedures.

If you do **not** use two-phase commit, do **not** use this command.

This command should be used only if the syncpoint manager has failed to resolve a transaction.

Syntax

```

▶▶ rsvmqtrn [-c] [-b] -m QMgrName Transaction ▶▶

```

Required parameters

-c Specifies a commit decision.

-b Specifies a backout decision.

There is no default; you must supply one of these options.

-m *QMgrName*

Specifies the name of the queue manager whose transactions are to be resolved. The queue manager name must be specified.

Transaction

Specifies the transaction number of the transaction of interest. The number can be determined by using the **dspmqtrn** command to display all transactions on a queue manager that have been left in a prepared (in-doubt) state.

rsvmqtrn

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

dspmqrn Display list of prepared transactions

runmqchi (Run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to the *MQSeries Distributed Queuing Guide*.

Syntax

```

▶▶—runmqchi [ -b ] [ -q InitiationQName ] [ -m QMgrName ]▶▶

```

Optional parameters

- b** The default is to run the command in the foreground. This flag runs the command in batch (background) operation. The default batch queue is SYS\$BATCH and you can override the default with the logical name MQS_BATCH.
- 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 160.

runmqchl (Run channel)

Purpose

Use the **runmqchl** command to run a Sender (SDR), a Requester (RQSTR), or a fully-qualified server (SVR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax

```

▶▶—runmqchl [ -b ] -c ChannelName [ -m QMgrName ]▶▶

```

Required parameters

-c *ChannelName*
 Specifies the name of the channel to run.

Optional parameters

-b The default is to run the command in the foreground. This flag runs the command in batch (background) operation. The default batch queue is SYS\$BATCH and you can override the default with the logical name MQS_BATCH.

-m *QMgrName*
 Specifies the name of the queue manager with which this channel is associated. If no name is specified, the default queue manager is used.

Return codes

0 Command completed normally
10 Command completed with unexpected results
20 An error occurred during processing

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages. You should also review the \$SYSTEM error log because problems that occur before the channel is associated with the queue manager are recorded there.

runmqdlq (Run dead-letter queue handler)

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, a utility that you can run to monitor and handle messages on a dead-letter queue.

The dead-letter queue handler can be used to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from SYS\$INPUT. When the command is processed, the results and a summary are put into a report that is sent to SYS\$OUTPUT.

By taking SYS\$INPUT 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.

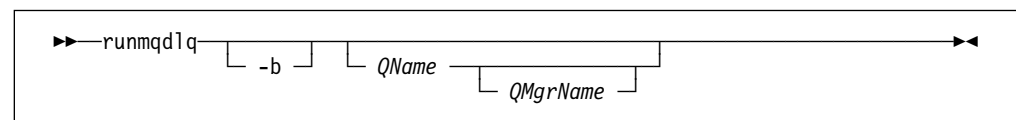
If the DLQ handler is used in foreground mode without redirecting SYS\$INPUT from a file, (the rules table) the DLQ handler:

- Reads its input from the keyboard.
- Does not start to process the named queue until it receives an end_of_file (ctrl-Z) character.

If used in batch mode SYS\$INPUT must be redirected from a file.

For more information about rules tables and how to construct them, see “The DLQ handler rules table” on page 108.

Syntax



Optional parameters

The MQSC rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName Specifies the name of the queue to be processed.

If no name is specified the dead letter queue defined for the local queue manager is used. If one or more blanks (' ') are used, the dead letter queue of the local queue manager is explicitly assigned.

A DLQ handler can be used to select particular messages on a dead-letter queue for special processing. For example, 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.

runmqdlq

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.

runmqsr (Run listener)

Purpose

The runmqsr (Run listener) command starts a listener batch job for inbound LU 6.2 channels.

The default is to run the command in the foreground. This flag runs the command in batch (background) operation. The default batch queue is SYS\$BATCH and you can override the default with the logical name MQS_BATCH.

Syntax

```
▶▶ runmqsr -g GatewayName (AccessName) -n TpName [-m QMgrName] ▶▶
```

Required parameters

-g *GatewayName (AccessName)*
Specifies the gateway name of the machine on which **runmqsr** is being run.

AccessName
Specifies the access name on the gateway that defines the LUs serviced by the listener.

-n *TpName*
Specifies the tpname.

Optional parameters

-m *QMgrName*
Specifies the name of the queue manager with which the listener is associated. If no name is specified, the command operates on the default queue manager.

Return codes

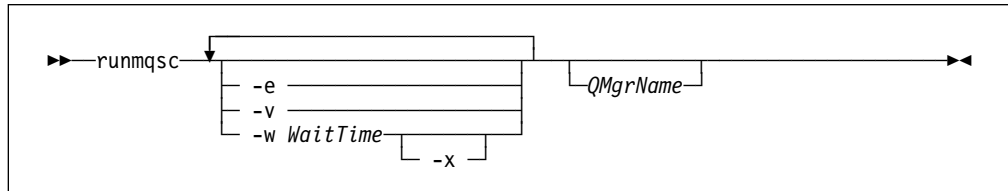
- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

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 the *MQSeries Command Reference*.

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 only available on a local queue manager.
- Direct mode** MQSC commands are sent directly to a local queue manager.
- Indirect mode** MQSC commands are run on a remote queue manager. These commands are put on the command queue on a remote queue manager and are run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The **runmqsc** command takes its input from `SYSD$INPUT`. When the commands are processed, the results and a summary are put into a report that is sent to `SYSD$OUTPUT`.

By taking `SYSD$INPUT` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

Optional parameters

- e** Prevents source text for the MQSC commands from being copied into a report. This is useful when you enter commands interactively.
- v** 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 73 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 MVS/ESA. This flag applies only in indirect mode. The -w flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the MQSeries for MVS/ESA command queue.

QMgrName

Specifies the name of the target queue manager on which the MQSC commands are to be run. If omitted, the MQSC commands run on the default queue manager.

Return codes

- 0 MQSC command file processed successfully.
- 10 MQSC command file processed with errors—report contains reasons for failing commands.
- 20 Error—MQSC command file not run.

Examples

1. Type in this command at the OpenVMS command prompt:

```
runmqsc
```

Now you can type MQSC commands directly at the OpenVMS command prompt. No queue manager name was specified, therefore, the MQSC commands are processed on the default queue manager.

2. Use this command to specify that MQSC commands are verified only:

```
runmqsc -v "BANK" < DKA0:[USERS]COMMFIL.IN
```

This verifies the MQSC command file `commfile.in` in directory `DKA0:[USERS]`. The queue manager name is `BANK`. The output is displayed in the current window.

3. This command runs the MQSC command file `MQS_ROOT:[MQM.MQSC]MQSCFILE.IN` against the default queue manager.

```
runmqsc < MQS_ROOT:[MQM.MQSC]MQSCFILE.IN > MQS_ROOT:[MQM.MQSC]MQSCFILE.OUT
```

In this example, the output is directed to file `MQS_ROOT:[MQM.MQSC]MQSCFILE.OUT`.

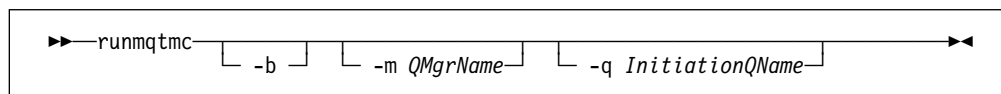
runmqtmc (Start client trigger monitor)

Purpose

Use the **runmqtmc** command to invoke a trigger monitor for a client. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide*.

Note: This command is available **only** on VMS, OS/2, and AIX clients.

Syntax



Optional parameters

- b** The default is to run the command in the foreground. This flag runs the command in batch (background) operation. The default batch queue is SYSS\$BATCH and you can override the default with the logical name MQS_BATCH.
- m** *QMgrName*
Specifies the name of the queue manager on which the client trigger monitor operates. If omitted, the client trigger monitor operates on the default queue manager.
- q** *InitiationQName*
Specifies the name of the initiation queue to be processed. If omitted, SYSTEM.DEFAULT.INITIATION.QUEUE is used.

Return codes

- 0** Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.
- 10** Client trigger monitor interrupted by an error.
- 20** Error—client trigger monitor not run.

runmqtrm (Start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide*.

Syntax

```

▶▶—runmqtrm [ -b ] [ -m QMgrName ] [ -q InitiationQName ]▶▶

```

Optional parameters

- b** The default is to run the command in the foreground. This flag runs the command in batch (background) operation. The default batch queue is SYS\$BATCH and you can override the default with the logical name MQS_BATCH.
- m** *QMgrName*
Specifies the name of the queue manager on which the trigger monitor operates. If omitted, the trigger monitor operates on the default queue manager.
- q** *InitiationQName*
Specifies the name of the initiation queue to be processed. If omitted, SYSTEM.DEFAULT.INITIATION.QUEUE is used.

Return codes

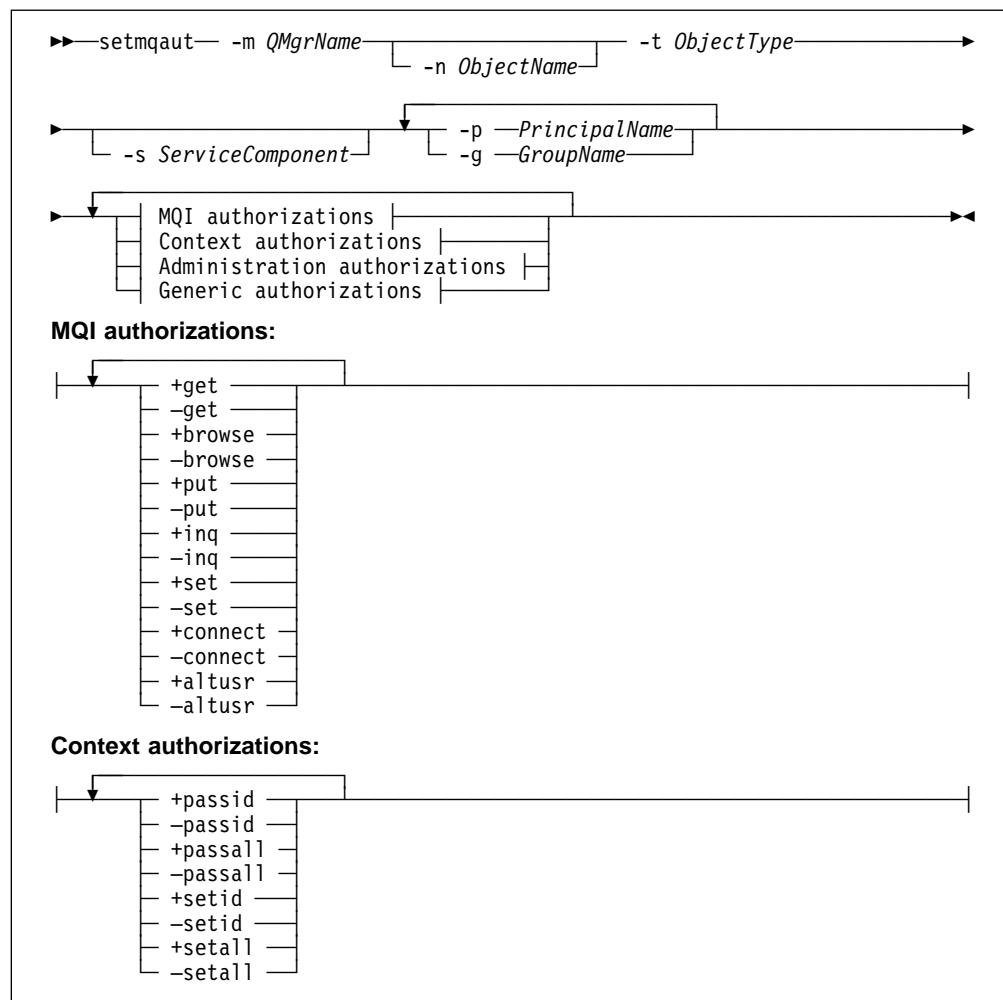
- 0** Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
- 10** Trigger monitor interrupted by an error.
- 20** Error—trigger monitor not run.

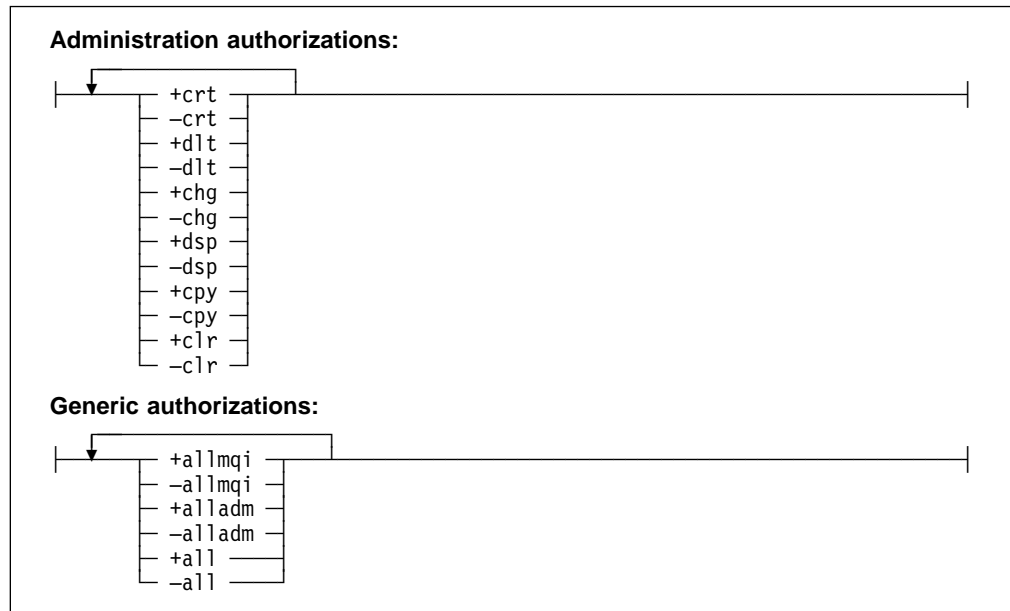
setmqaut (Set/reset authority)

Purpose

Use the **setmqaut** command to change the authorizations to an object or to a class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

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.

Attention: 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

-m *QMgrName*

Specifies the name of the queue manager of the object for which the authorizations are to be changed. The name can contain up to 48 characters.

-t *ObjectType*

Specifies the type of object for which the authorizations are to be changed.

Possible values are:

- **q** or **queue**
- **prcs** or **process**
- **qmgr**

Optional parameters

-n *ObjectName*

Specifies the name of the object for which the authorizations are to be changed.

This is a required parameter **unless** it is the queue manager itself. You must specify the name of a queue manager, queue, or process, but must not use a generic name.

-p *PrincipalName*

Specifies the name of the principal for which the authorizations are to be changed.

You must have at least one principal or one group.

-g *GroupName*

Specifies the name of the rights identifier representing the user group whose authorizations are to be changed. You can specify more than one rights identifier 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 13 on page 214 shows the authorities that can be given to the different object types.

Table 13. Specifying authorizations for different object types			
Authority	Queue	Process	Qmgr
all	√	√	√
alladm	√	√	√
allmqi	√	√	√
altusr			√
browse	√		
chg	√	√	√
clr	√		
connect			√
crt	√	√	√
dlt	√	√	√
dsp	√	√	√
put	√		
inq	√	√	√
get	√		
passall	√		
passid	√		
set	√	√	√
setall	√		√
setid	√		√

Authorizations for MQI calls

- altusr** Use an alternate user ID in a message.
See the *MQSeries Application Programming Guide* for more information about alternate user IDs.
- 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).
cpy	Copy the attributes of the specified object (PCF Copy commands 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

setmqaut

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g tango +inq +alladm
```

The authorizations are being given to user group tango and the associated authorization list specifies that user group tango:

- Can issue MQINQ calls.
 - Has authority to perform all administration operations on that object.
2. In this example, the authorization list specifies that user group foxy:
 - Cannot issue any calls from the MQI to the specified queue.
 - Has authority to perform all administration operations on the specified queue.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g foxy -allmqi +alladm
```

Related commands

dspmqaut Display authority

strmqcsv (Start command server)

Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables MQSeries to process commands sent to the command queue.

Syntax

```
▶▶—strmqcsv—QMgrName—◀◀
```

Required 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

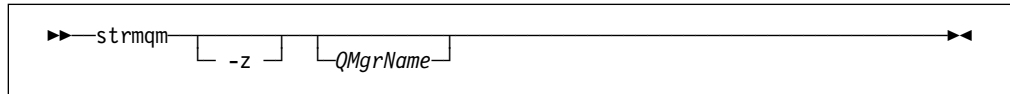
- endmqcsv** End a command server
- dspmqcsv** Display the status of a command server

strmqm (Start queue manager)

Purpose

Use the **strmqm** command to start a local queue manager.

Syntax



Optional parameters

QMgrName

Specifies the name of a local queue manager to be started. If omitted, the default queue manager is started.

-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.

Return codes

0	Queue manager started
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
23	Log not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid

Examples

The following command starts the queue manager account:

```
strmqm account
```

Related commands

crtmqm	Create a queue manager
dltmqm	Delete a queue manager
endmqm	End a queue manager

strmqtrc (Start MQSeries trace)

Purpose

Use the **strmqtrc** command to enable tracing. This command can be run whether tracing is enabled or not. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

Syntax

```

▶▶ strmqtrc [ -m QMgrName ] [ -e ] [ -t TraceType ] ▶▶

```

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.

- e** If this flag is specified, early tracing is requested. Consequently, it is possible to trace the creation or startup of a queue manager. This involves trace information being written, before the processes know to which MQSeries component they belong. Any process, belonging to any component of any queue manager, traces its early processing if this flag is specified. The default, if this flag is not specified, is not to perform early tracing.

-t *TraceType*

Defines which points during processing can be traced. If this flag is omitted, all trace points are enabled and a full trace generated.

Alternatively, one or more of the options in the following list can be supplied.

Note: If multiple trace types are supplied, each *must* have its own -t flag. Any number of -t flags can be specified, as long as each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

- all** Output data for every trace point in the system. This is also the default if the -t flag is not specified.

- api** Output data for trace points associated with the MQI and major queue manager components.

comms

Output data for trace points associated with data flowing over communications networks.

csflows

Output data for trace points associated with processing flow in common services.

lqmflows

Output data for trace points associated with processing flow in the local queue manager.

remoteflows

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

otherflows

Output data for trace points associated with processing flow in other components.

csdata

Output data for trace points associated with internal data buffers in common services.

lqmdata

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

remotedata

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

otherdata

Output data for trace points associated with internal data buffers in other components.

versiondata

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

commentary

Output data for trace points associated with comments in the MQSeries components.

Return codes

AMQ7024

This message is issued if arguments that are not valid are supplied to the command.

AMQ8304

The maximum number of nine concurrent traces is already running.

Examples

This command enables tracing of data from common services and the local queue manager, for a queue manager called QM1.

```
strmqtrc -m QM1 -t csdata -t lqmdata
```

Related commands

- dspmqtrc** Display formatted trace output
- endmqtrc** End MQSeries trace

strmqtrc

Part 3. Appendixes

Appendix A. MQSeries for Digital OpenVMS at a glance

Program and part number

- 5697-271 MQSeries for Digital OpenVMS VAX Version 2 Release 2 (Europe, the Middle East and Africa only), part numbers 83H8438 for TK50 tape and 83H8440 for CD-ROM.
- 5697-270 MQSeries for Digital OpenVMS AXP Version 2 Release 2 (Europe, the Middle East and Africa only), part number 83H8439.

Hardware requirements

- MQSeries Servers:

Any Digital VAX or AXP machine

Minimum system disk space 16 MB on a VAX machine and 18 MB on an AXP machine.

Software requirements

Software requirements are identical for server and client Digital OpenVMS environments unless otherwise stated.

Minimum supported levels are shown. Later levels, if any, will be supported unless otherwise stated.

- Digital OpenVMS Version 6.2 or later
- MQSeries Clients:

Client code for Digital OpenVMS, OS/2, DOS and Windows 3.1 workstations is distributed with the server code.

The Windows 3.1 client can operate under Windows 3.1, Windows 95 or within the WIN-OS/2 environment under OS/2.

Client software provides a remote interface to a LAN server. It may reside at the server or at a file server and be dynamically copied to the client for use, or it may reside on the client disk space.

Client support does not result in distributed coordination of units of work.

Connectivity

Network protocols supported are SNA LU6.2, TCP/IP, DECnet Phase IV, and DECnet Phase V.

For SNA connectivity:

SNA APPC LU6.2 Programming Interface software and license must be installed, and access to a suitably configured DECnet SNA gateway.

For TCP/IP connectivity:

DEC TCP/IP Services for OpenVMS Version 4.0 or higher or
Cisco Multinet Version 3.5 or higher

Overview

Compilers supported for MQSeries for Digital OpenVMS applications

- Programs can be written using C, C++ or COBOL
- C programs can use the DEC C compiler
- C++ programs can use the DEC C++ compiler
- COBOL programs can use the DEC COBOL compiler

Delivery

CD-ROM containing MQSeries for Digital OpenVMS installation images.

MQSeries for OpenVMS VAX is also available on TK50 tape.

Installation

MQSeries for Digital OpenVMS is installed with the OpenVMS VMSINSTAL utility. Installation takes approximately 10 minutes.

Appendix B. System defaults

The sample MQSC command file `amqscoma.tst` contains definitions for the MQSeries for Digital OpenVMS default and system objects. The default object definitions contain a complete set of attributes for that object. When you create an object, its attributes are inherited from the default object, except the ones you explicitly specify. The system objects are required for the operation of a queue manager or channel. Table 14 lists the objects defined in `amqscoma.tst`.

You should create these objects for each queue manager on a given node. To create these objects, see “Running the supplied MQSC command files” on page 55.

<i>Table 14. Objects included in amqscoma.tst</i>	
Object name	Description
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.DEAD.LETTER.QUEUE	Sample dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.PROCESS	Default process definition.
SYSTEM.DEF.SENDER	Default sender channel.
SYSTEM.DEF.SERVER	Default server channel.
SYSTEM.DEF.RECEIVER	Default receiver channel.
SYSTEM.DEF.REQUESTER	Default requester channel.
SYSTEM.DEF.SVRCONN	Default server connection channel.
SYSTEM.DEF.CLNTCONN	Default client connection channel.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	Channel synchronization queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands, and PCF commands.
SYSTEM.MQSC.REPLY.QUEUE	MQSC reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channel events.

Defaults

Appendix C. Directory structure

Figure 16 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.

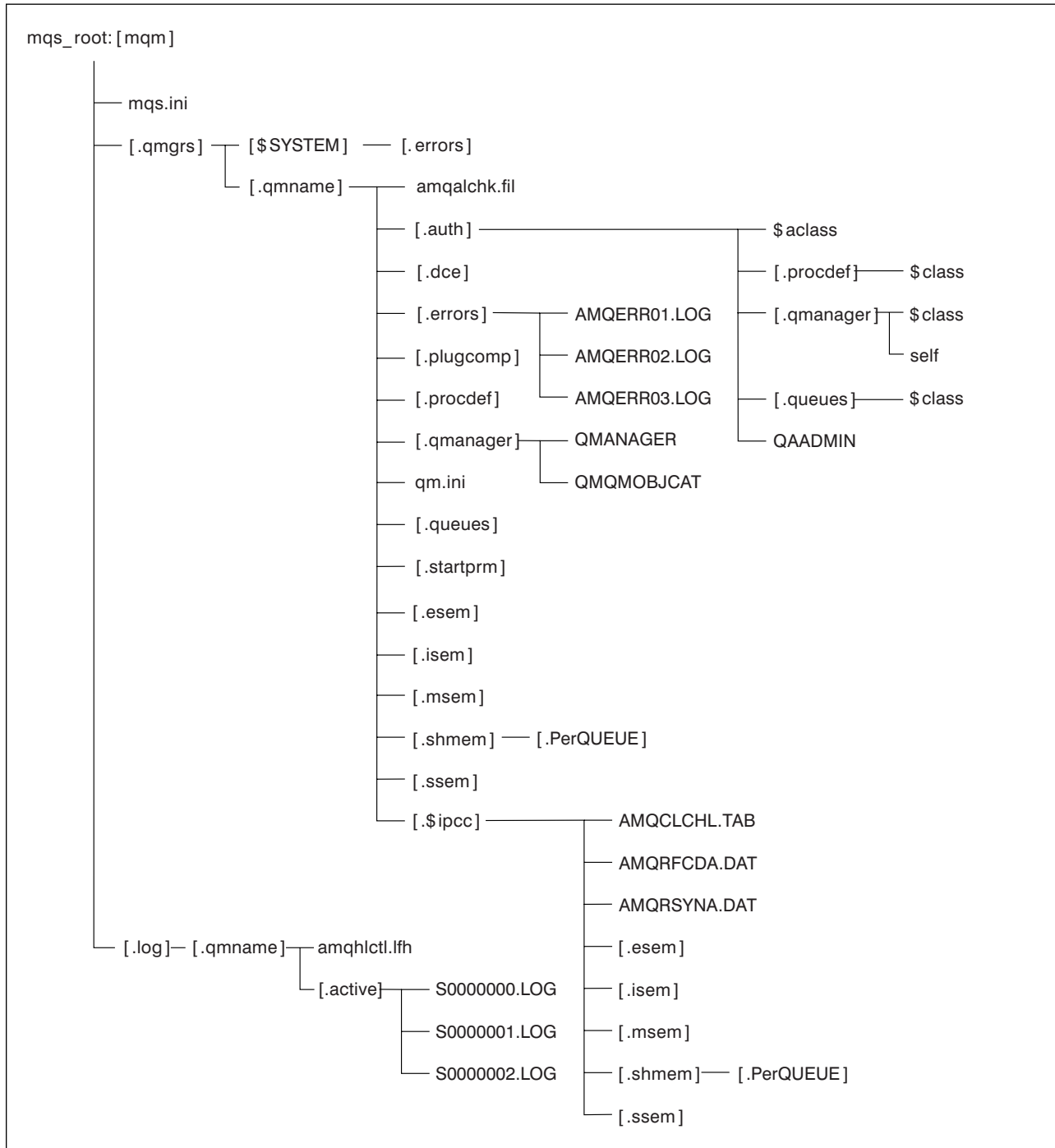


Figure 16. Default directory structure after a queue manager has been started

Directory structure

In Figure 16, 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 `MQS_ROOT:[MQM.QMGRS.QMNAME]`.

AMQALCHK.FIL

Checkpoint file containing information about last checkpoint.

AUTH.

This directory contains subdirectories and files associated with authority.

\$ACCLASS This file contains the authority stanzas for all classes.

PROCDEF This directory contains a file for each process definition. Each file contains the authority stanzas for the associated process definition.

\$CLASS This file contains the authority stanzas for the process definition class.

QMANAGER

\$CLASS This file contains the authority stanzas for the queue manager class.

SELF This file contains the authority stanzas for the queue manager object.

QUEUES This directory contains a file for each queue. Each file contains the authority stanzas for the associated queue.

\$CLASS This file contains the authority stanzas for the queue class.

QAADMIN File used internally for controlling authorizations.

DCE

Empty directory reserved for use by DCE support.

ERRORS

The operator message files, from newest to oldest:

AMQERR01.LOG

AMQERR02.LOG

AMQERR03.LOG

PLUGCOMP

Empty directory reserved for use by installable services.

PROCDEF

Each MQSeries process definition is associated with a file in this directory. The file name matches the process definition name—subject to certain restrictions; see “Understanding MQSeries file names” on page 42.

QMANAGER

QMANAGER The queue manager object.

QMOMOBJCAT 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 42.
STARTPRM	Directory containing temporary files used internally.
ESEM	Directories containing files used internally.
ISEM	
MSEM	
SHMEM	
PerQUEUE	
SSEM	Directory containing files used internally.
\$IPCC	
AMQCLCHL.TAB	Client channel table file.
AMQRFCDA.DAT	Channel table file.
AMQRSYNA.DAT	Channel synchronization file.
ESEM	Directories containing files used internally.
MSEM	
SSEM	
ISEM	
SHMEM	
	PerQUEUE Directory containing files used internally.

Queue manager log directory structure

By default, the following directories and files are found in MQS_ROOT:[MQM.LOG.qmname].

The following subdirectories and files exist after you have installed MQSeries, created and started a queue manager, and have been using that queue manager for some time.

AMQHLCTL.LFH	Log control file.
ACTIVE	This directory contains the log files, numbered as follows: S0000000.LOG S0000001.LOG S0000002.LOG ... and so on.

Directory structure

Appendix D. Sample MQI programs and MQSC files

MQSeries for Digital OpenVMS provides a set of short sample MQI programs and MQSC command files that you can use and experiment with.

MQSC command file samples

Table 15 lists the MQSC command file samples. These are simply ASCII text files containing MQSC commands. You can invoke the **runmqsc** command against each file in turn to create the objects specified in the file. See “Running the supplied MQSC command files” on page 55.

By default, these files are located in directory MQS_EXAMPLES.

<i>Table 15. MQSC command files</i>	
File name	Purpose
AMQSCOMA.TST	Contains definitions of the default and system objects. These are required. Any object you define inherits attributes from the default objects except the attributes that you specify. The system objects are support the operation of a queue manager.
AMQSCOS0.TST	Creates a set of MQI objects for use with the C and COBOL program samples.

C and COBOL program samples

Table 16 lists the sample MQI source files. By default, the source files are located in directory MQS_EXAMPLES and the compiled versions in directory [BIN] under MQS_EXAMPLES. To find out more about what the programs do and how to use them, see the *MQSeries Application Programming Guide*.

<i>Table 16 (Page 1 of 2). Sample programs - source files</i>		
C	COBOL	Purpose
AMQSBG0.C	–	Reads and then outputs both the message descriptor and message context fields of all the messages on a specified queue.
AMQSECHA.C	AMQVECHX.COB	Echoes a message from a message queue to the reply-to queue. Can be run as a triggered application program.
AMQSGBR0.C	AMQ0GBR0.COB	Writes messages from a queue to SYS\$OUTPUT leaving the messages on the queue. Uses MQGET with the browse option.
AMQSGET0.C	AMQ0GET0.COB	Removes the messages from the named queue (using MQGET) and writes them to SYS\$OUTPUT
AMQSINQA.C	AMQVINQX.COB	Reads the triggered queue; each request read as a queue name; responds with information about that queue.
AMQSPUT0.C	AMQ0PUT0.COB	Copies SYS\$INPUT to a message and then puts this message on a specified queue.
AMQSREQ0.C	AMQ0REQ0.COB	Puts request messages on a specified queue and then displays the reply messages.
AMQSSETA.C	AMQVSETX.COB	Inhibits puts on a named queue and responds with a statement of the result. Runs as a triggered application.

Samples

<i>Table 16 (Page 2 of 2). Sample programs - source files</i>		
C	COBOL	Purpose
AMQSTRG0.C	–	A trigger monitor that reads a named initiation queue and then starts the program associated with each trigger message. Provides a subset of the full triggering function of the supplied runmqtrm command.
AMQSVFCX.C	–	A sample C skeleton of a Data Conversion exit routine.
Note: You can create the objects required by these samples using the MQSC command file amqscos0.tst.		

Miscellaneous tools

These tool files are provided to support the formatter and code conversion.

<i>Table 17. Miscellaneous files</i>		
File name	Location	Purpose
AMQTRC.FMT	SYS\$LIBRARY	Defines MQSeries trace formats.
CCSID.TBL	MQS_ROOT:[MQM.CONV.TABLE]	Edit this file to add any newly supported CCSID values to your MQSeries system. For more information about CCSID, see the CDRA documentation.

Appendix E. Codeset support on MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS supports most of the codesets used by the locales – that is, the subsets of the user's environment which define the conventions for a specific culture – that are provided as standard on MQSeries for Digital OpenVMS.

If the locale is not set the CCSID used is 819 - the ISO8859-1 codeset.

The CCSID (Coded Character Set Identifier) used in MQSeries to identify the codeset used for the message and message header data is obtained by analyzing the LC_CTYPE category of the locale configuration.

Table 18 shows the locales and the CCSIDs that are registered for the codeset used by the locale.

<i>Table 18 (Page 1 of 2). Locales and CCSIDs</i>			
Locale	Language	codeset	CCSID
C	English	ISO8859-1	819
CS_CZ_ISO8859-2	Czech	ISO8859-2	912
DA_DK_ISO8859-1	Danish	ISO8859-1	819
DE_DE_ISO8859-1	German	ISO8859-1	819
DE_CH_ISO8859-1	German - Switzerland	ISO8859-1	819
EL_GR_ISO8859-7	Greek	ISO8859-7	813
EN_GB_ISO8859-1	English - United Kingdom	ISO8859-1	819
EN_US_ISO8859-1	English - USA	ISO8859-1	819
ES_ES_ISO8859-1	Spanish	ISO8859-1	819
FI_FI_ISO8859-1	Finnish	ISO8859-1	819
FR_FR_ISO8859-1	French - France	ISO8859-1	819
FR_BE_ISO8859-1	French - Belgium	ISO8859-1	819
FR_CA_ISO8859-1	French - Canada	ISO8859-1	819
FR_CH_ISO8859-1	French - Switzerland	ISO8859-1	819
HU_HU_ISO8859-2	Hungarian	ISO8859-2	912
IS_IS_ISO8859-1	Icelandic	ISO8859-1	819
IT_IT_ISO8859-1	Italian - Italy	ISO8859-1	819
IW_IL_ISO8859-8	Hebrew	ISO8859-8	916
JA_JP_EUCJP	Japanese	eucJP	954
JA_JP_SDECKANJI	Japanese	SDECKANJI	954**
JA_JP_SJIS	Japanese	SJIS	932
KO_KR_DECKOREAN	Korean	DECKOREAN	970**
NL_NL_ISO8859-1	Dutch - Netherlands	ISO8859-1	819
NL_BE_ISO8859-1	Dutch - Belgium	ISO8859-1	819
NO_NO_ISO8859-1	Norwegian	ISO8859-1	819
PL_PL_ISO8859-2	Polish	ISO8859-2	912
PT_PT_ISO8859-1	Portuguese	ISO8859-1	819

Supported codesets

<i>Table 18 (Page 2 of 2). Locales and CCSIDs</i>			
Locale	Language	codeset	CCSID
SK_SK_ISO8859-2	Slovak	ISO8859-2	912
RU_RU_ISO8859-5	Cyrillic	ISO8859-5	915
SV_SE_ISO8859-1	Swedish	ISO8859-1	819
TR_TR_ISO8859-9	Turkish	ISO8859-9	920
ZH_CN_DECHANZI ZH_HK_DECHANZI	Chinese - Simplified Chinese - Simplified	DECHANZI DECHANZI	1383** 1383**
ZH_HK_EUCTW ZH_TW_EUCTW ZH_HK_DECHANYU ZH_TW_DECHANYU ZH_HK_BIG5 ZH_TW_BIG5	Chinese - Traditional Chinese - Traditional Chinese - Traditional Chinese - Traditional Chinese - Traditional Chinese - Traditional	eucTW eucTW DECHANYU DECHANYU big5 big5	964 964 964** 964** 950 950
Note:			
** The CCSID used is the nearest registered IBM CCSID.			

For further information listing inter-platform support for these locales, see the *MQSeries Application Programming Reference* manual.

Appendix F. Fast messages and trusted applications.

Note

The information in this appendix will be inserted into the *MQSeries Application Programming Guide* the next time that this book is refreshed.

Fast messages

Fast messages are nonpersistent messages sent by channels outside of batches of persistent messages.

The performance implication is that the batch size can be set to a large number to efficiently move the persistent messages, while the nonpersistent messages do not wait for batch completion. In a lightly loaded system, nonpersistent messages can travel between applications 80% faster than normal-speed channels.

To enable fast messages on a particular channel of type sender, server, receiver, or requester, set the following definitions at both ends of the channel **after** the CHLTYPE line:

```
DESCR('>>> whatever') +
```

The '>>>' as the first characters in the channel description defines this channel as fast for nonpersistent messages.

Note: If the other end of the channel does not support the option, the channel runs at normal speed.

Trusted applications

Applications that have been shown to behave properly can be defined as "Trusted" and will not need an agent process to interface with the queue manager.

Therefore, the process AMQZLAO will not exist for these applications. These applications become an extension to the queue manager and there is no protection from them overwriting queue manager storage.

The time to process MQPUT and MQGET calls of nonpersistent messages is reduced by 75%. The MQSeries channel programs can be defined as "Trusted".

Setting up "trusted" applications

Trusted applications connect directly to queue manager resources, and may need to create certain resources like shared memory.

The resources can be modified or deleted by another queue manager process and, therefore, **must** be owned by the same UIC.

The queue manager processes all run under the MQS_SERVER UIC, and so trusted applications **must** also run under this UIC.

Trusted applications

An application runs in “trusted” mode if the logical name MQ_CONNECT_TYPE is defined as FAST.

To set up a trusted application, carry out the following procedure:

1. Run your application from the MQS_SERVER UIC. Either, login to the MQM account, or issue the following command:

```
$ SET UIC MQS_SERVER
```

2. Define the logical name MQ_CONNECT_TYPE as FAST, so that your application program can access the server, as follows:

```
$ DEFINE MQ_CONNECT_TYPE FAST
```

Note: FAST is case sensitive.

Your application will now run as a trusted application.

Trusted channel programs

Channel programs started using the **runmqsc** start channel command run under the MQS_SERVER UIC. Channel receiver programs started by incoming TCP (or DECnet connect) run under the MQS_SERVER UIC, because the definition of the TCP service (or DECnet object) specifies that the process is to run under the MQM account.

If the **runmqchl** command is used to start channels you **must** either login to the MQM account, or issue the command:

```
$ set uic MQS_SERVER
```

so that the channel program runs under the MQS_SERVER UIC.

To set up a trusted channel:

1. Define the logical name MQ_CONNECT_TYPE to be FAST, so that the channel program can access the logical.

If you want to have all channels run as trusted applications, you can define the logical name in the group table for the MQS_SERVER UIC.

The group for the MQS_SERVER UIC defaults to 400 during installation of MQSeries.

For example:

```
$ DEFINE/TABLE=LNMSGROUP_000400 MQ_CONNECT_TYPE FAST
```

Note: All applications running in the same group as the MQS_SERVER UIC will attempt to run as trusted applications. If they are not running under the MQS_SERVER UIC they will fail when they attempt to connect to the queue manager.

2. Start your channels using the **runmqsc** start channel command. This starts the channel processes under the MQS_SERVER UIC. The channel programs will find the connection type to be **FAST** from the logical defined in the group logical name table.
3. If you use the **runmqchl** command to start your channels, login to the MQM account, or issue the command:

```
$ set uic MQS_SERVER
```

Note: If you use the `-b` flag to indicate that the channel program should be submitted as a batch job, you **must** login to the MQM account because the batch job runs under the account that submitted it.

4. For SNA listener processes, started using the `runmqlsr` command, you **must** also login to the MQM account, because the listener process runs as a batch job under the account that submitted it.

Restrictions on Trusted Applications

There are restrictions to the following when using trusted applications:

- Threaded applications
- Security
- Usage
- Fast channels

Threaded applications

On Digital OVMS, trusted applications cannot be threaded.

Security

Trusted applications run MQSeries code directly and use resources owned by `MQS_SERVER`, and so **must** run under the `MQS_SERVER` UIC.

Usage

You **must** stop trusted applications before ending the queue manager with the `endmqm` command. This is because Trusted applications are part of the MQSeries environment.

Unless the application is ended, MQSeries resources will not be freed and the queue manager cannot be restarted.

If the application has been started under a system service and MQSeries queue managers are being started automatically, it is necessary to end the application before shutting the operating system down, because the order of process termination cannot be guaranteed.

For this reason, when the queue manager is ended with the command `endmqm -p <queue manager>`, the queue manager ends trusted applications that are still connected to the queue manager before closing.

Fast Channels

You should not stop channels running as trusted applications with mode `FORCE`. If you do, the queue manager may be left in an undefined state.

Building applications

On Digital OVMS there is a limited number of shared memory segments. MQSeries uses two additional shared memory segments for trusted applications, which reduces the amount of shared storage available to Trusted MQSeries applications.

Integrity

Before you make an application run as a trusted application check that the application itself, as well as the user exits and installable services are behaving properly.

For example, when making channel programs run Trusted, all the channel exits must also be checked to make sure they behave properly.

Appendix G. Code page conversion tables

Note

The information in this appendix will be inserted into the *MQSeries Application Programming Reference* the next time that this book is refreshed.

Each of the tables shows the conversion support for the characters used by one language.

Some of the coded character set identifiers (CCSIDs) are used by many languages, for example CCSID 819 (ISO8859-1 Western European), and appear in many tables. Other CCSIDs, for example CCSID 273 (German EBCDIC), appear in only one table.

The following terms are used in the tables:

ISO	Indicates that the CCSID is for an ISO 8859 codeset
pc-A	Indicates in the AIX and GIS rows that the CCSID is an IBM defined CCSID used in AIX, AT&T, and OS/2.
-8	Indicates in the HP-UX rows that the CCSID is for the HP-UX defined codeset <i>roman8</i>
GIS	Indicates MQSeries for AT&T GIS UNIX
NT	Indicates MQSeries for Windows NT
Solaris	Indicates MQSeries for Sun Solaris
SunOS	Indicates MQSeries for SunOS
SINIX, DC/OSx	Indicates MQSeries for SINIX and DC/OSx
DEC OpenVMS	Indicates MQSeries for Digital OpenVMS and MQSeries for Digital VMS VAX
Tandem	Indicates MQSeries for Tandem NonStop Kernel V2.2

The following codes are used in the tables:

Y	Conversion at target supported going to and from source
y	No conversion is required because the different MQSeries products are operating in the same CCSID

The default for data conversion is for the conversion to be performed at the target (receiving) system.

Where a cell in a table is blank, conversion is not supported by the target product.

If the source product supports the conversion a channel can be set up and data exchanged by setting the channel attribute **DataConversion** to YES at the source. To determine if the source product supports the conversion, read the relevant table with source and target reversed. If conversion is shown as supported, it is possible to do conversion in the source product.

Code page conversion tables

Notes:

1. If you have MQSeries for MVS/ESA V1.1.3 and have installed APAR PN73611, you can change the default CCSID. If you have an earlier release, or have not applied this APAR, CCSID 500 is always used; this means that you can only use the multilingual code page (Table 27 on page 251).
2. Conversion for MQSeries client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.
3. The OS/2 and Solaris rows include information from some country specific versions. Not all of the conversions shown in the OS/2 and Solaris rows are supported by all OS/2 and Solaris versions.

For an extended list of CCSIDs, see the *Character Data Representation Reference*. See Table 43 on page 267 for a cross reference between some of the CCSID numbers and some industry codeset names.

MQSeries for MVS/ESA V1.1.4 provides conversions between single byte CCSIDs in addition to those listed in the language tables. A complete list of conversions provided is shown in Table 44 on page 268.

Code page conversion tables

<i>Table 19. Conversion support: US ENGLISH</i>						
Target ▼	Source→	MVS, OS/400	OS/2, GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	AIX, GIS, NT	HP-UX
	CCSID	37	437	819	850	1051
MVS	37	y	Y†	Y†	Y†	Y†
OS/400	37	y	Y	Y	Y	
OS/2	437	Y	y		Y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Code page conversion tables

<i>Table 20. Conversion support: GERMAN</i>						
Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	273	437	819	850	1051
MVS	273	y	Y†	Y†	Y†	Y†
OS/400	273	y	Y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Target ▼	Source→	MVS, OS/400	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	OS/2, GIS, NT	HP-UX
	CCSID	277	819	850	865	1051
MVS	277	y	Y†	Y†	Y†	Y†
OS/400	277	y	Y	Y	Y	
OS/2	850	Y		y	Y	
OS/2	865	Y		Y	y	
AIX (pc-A)	850	Y	Y	y		Y*
AIX (ISO)	819	Y	y	Y		Y*
HP-UX (ISO)	819	Y	y	Y	Y	Y
HP-UX (-8)	1051	Y	Y	Y		y
GIS (ISO)	819	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	y	Y	Y
GIS (pc-A)	865	Y	Y	Y	y	
NT	850	Y	Y	y	Y	Y
NT	865	Y	Y	Y	y	
Solaris	819	Y	y	Y	Y	Y
SunOS	819	Y	y	Y		
SINIX, DC/OSx	819	Y	y	Y	Y	Y
DEC-OVMS	819	Y	y	Y	Y	Y
Tandem	819	Y	y	Y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Code page conversion tables

Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	OS/2, NT	HP-UX
	CCSID	278	437	819	850	865	1051
MVS	278	y	Y†	Y†	Y†	Y†	Y†
OS/400	278	y	Y	Y	Y	Y	
OS/2	850	Y	Y		y	Y	
OS/2	865	Y	Y		Y	Y	
AIX (pc-A)	850	Y	Y	Y	y		Y*
AIX (ISO)	819	Y	Y*	y	Y		Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y		y
GIS (ISO)	819	Y	Y	y	Y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y	Y
NT	437	Y	y	Y	Y	Y	Y
NT	850	Y	Y	Y	y	Y	Y
NT	865	Y	Y	Y	Y	y	
Solaris	819	Y	Y	y	Y	Y	Y
SunOS	819	Y	Y	y	Y		
SINIX, DC/OSx	819	Y	Y	y	Y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y	Y
Tandem	819	Y	Y	y	Y	Y	Y
Note:							
* Supported on MQSeries for AIX version 2.2.1 or later.							
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.							

<i>Table 23. Conversion support: ITALIAN</i>						
Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	280	437	819	850	1051
MVS	280	y	Y†	Y†	Y†	Y†
OS/400	280	y	Y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Code page conversion tables

<i>Table 24. Conversion support: SPANISH</i>						
Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	284	437	819	850	1051
MVS	284	y	Y†	Y†	Y†	Y†
OS/400	284	y	Y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

<i>Table 25. Conversion support: UK ENGLISH / GAELIC</i>						
Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	285	437	819	850	1051
MVS	285	y	Y†	Y†	Y†	Y†
OS/400	285	y	Y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Code page conversion tables

<i>Table 26. Conversion support: FRENCH</i>						
Target ▼	Source→	MVS, OS/400	GIS, NT	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	297	437	819	850	1051
MVS	297	y	Y†	Y†	Y†	Y†
OS/400	297	y	Y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y	Y*	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	Y	y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Target ▼	Source→	GIS, NT	MVS, OS/400	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, AIX, GIS, NT	HP-UX
	CCSID	437	500	819	850	1051
MVS	500	Y†	y	Y†	Y†	Y†
OS/400	500	Y	y	Y	Y	
OS/2	850	Y	Y		y	
AIX (pc-A)	850	Y	Y	Y	y	Y*
AIX (ISO)	819	Y*	Y	y	Y	Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y	y
GIS (ISO)	819	Y	Y	y	Y	Y
GIS (pc-A)	437	y	Y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y
NT	437	y	Y	Y	Y	Y
NT	850	Y	Y	Y	y	Y
Solaris	819	Y	Y	y	Y	Y
SunOS	819	Y	Y	y	Y	
SINIX, DC/OSx	819	Y	Y	y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y
Tandem	819	Y	Y	y	Y	Y

Note:

* Supported on MQSeries for AIX version 2.2.1 or later.
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.

Code page conversion tables

Target ▼	Source→	OS/400	MVS, OS/400	AIX, HP-UX, GIS, Solaris, SunOS, DEC- OVMS DEC- OVMS	OS/2, AIX, GIS, NT	OS/2, GIS, NT	HP-UX
	CCSID	37	500	819	850	860	1051
MVS	500	Y†	y	Y†	Y†	Y†	Y†
OS/400	37	y	Y	Y	Y	Y	
OS/400	500	Y	y	Y	Y	Y	
OS/2	850	Y	Y		y	Y	
OS/2	860	Y	Y		Y	y	
AIX (pc-A)	850	Y	Y	Y	y		Y*
AIX (ISO)	819	Y	Y	y	Y		Y*
HP-UX (ISO)	819	Y	Y	y	Y	Y	Y
HP-UX (-8)	1051	Y	Y	Y	Y		y
GIS (ISO)	819	Y	Y	y	Y	Y	Y
GIS (pc-A)	850	Y	Y	Y	y	Y	Y
GIS (pc-A)	860	Y	Y	Y	Y	y	
NT	850	Y	Y	Y	y	Y	Y
NT	860	Y	Y	Y	Y	y	
Solaris	819	Y	Y	y	Y	Y	Y
SunOS	819	Y	Y	y	Y		
SINIX, DC/OSx	819	Y	Y	y	Y	Y	Y
DEC-OVMS	819	Y	Y	y	Y	Y	Y
Tandem	819	Y	Y	y	Y	Y	Y
Note:							
* Supported on MQSeries for AIX version 2.2.1 or later.							
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.							

<i>Table 29. Conversion support: ICELANDIC</i>						
Target ▼	Source→	AIX, HP-UX, GIS, Solaris, SunOS, DEC-OVMS Tandem	OS/2, GIS, NT	OS/2, AIX, NT	MVS, OS/400	HP-UX
	CCSID	819	850	861	871	1051
MVS	871	Y†	Y†	Y†	y	Y†
OS/400	871	Y	Y	Y	y	
OS/2	850		y	Y	Y	
OS/2	861		Y	y	Y	
AIX (pc-A)	850	Y	y		Y	Y*
AIX (ISO)	819	y	Y		Y	Y*
HP-UX (ISO)	819	y	Y	Y	Y	Y
HP-UX (-8)	1051	Y	Y		Y	y
GIS (ISO)	819	y	Y	Y	Y	Y
GIS (pc-A)	850	Y	y	Y	Y	Y
NT	850	Y	y	Y	Y	Y
NT	861	Y	Y	y	Y	
Solaris	819	y	Y	Y	Y	Y
SunOS	819	y	Y		Y	
SINIX, DC/OSx	819	y	Y	Y	Y	Y
DEC-OVMS	819	y	Y	Y	Y	Y
Tandem	819	y	Y	Y	Y	Y
Note:						
* Supported on MQSeries for AIX version 2.2.1 or later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						

Code page conversion tables

<i>Table 30. Conversion support: EASTERN EUROPEAN Languages</i>				
Target ↓ ▼	Source→	OS/2, NT	MVS, OS/400	AIX, HP-UX, GIS, Solaris, DEC-OVMS Tandem
	CCSID	852	870	912
MVS	870	Y†	y	Y†
OS/400	870	Y	y	Y
OS/2	852	y	Y	
AIX (ISO)	912	Y*	Y*	y
HP-UX (ISO)	912	Y	Y	y
GIS (ISO)	912	Y	Y	y
NT	852	y	Y	Y
Solaris	912	Y	Y	y
SunOS				
SINIX, DC/OSx	912	Y	Y	y
DEC-OVMS	912	Y	Y	y
Tandem	912	Y	Y	y
Note: The typical languages which use these CCSIDS include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovakian, and Sloven.				
Note:				
* Only on AIX V4.1 and later.				
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.				

<i>Table 31. Conversion support: CYRILLIC</i>						
Target ▼	Source→	OS/2, NT	OS/2, NT	OS/400	AIX, HP-UX, GIS, Solaris, DEC-OVMS Tandem	MVS, OS/400
	CCSID	855	866	880	915	1025
MVS	1025	Y†	Y†	Y†	Y†	y
OS/400	880			y	Y	Y
OS/400	1025	Y	Y	Y	Y	y
OS/2	855	y	Y		Y	Y
OS/2	866	Y	y		Y	Y
AIX (ISO)	915	Y+	Y*	Y+	y	Y+
HP-UX (ISO)	915	Y	Y**	Y	y	Y
GIS (ISO)	915	Y	Y	Y	y	Y
NT	855	y	Y	Y	Y	Y
NT	866	Y	y	Y	Y	Y
Solaris	915	Y	Y	Y	y	Y
SunOS						
SINIX, DC/OSx	915	Y	Y	Y	y	Y
DEC-OVMS	915	Y	Y	Y	y	Y
Tandem	915	Y	Y	Y	y	Y
Note: The typical languages which use these CCSIDS include Byelorussia (Belarus), Bulgarian, Macedonian, Russian, and Serbian.						
Note:						
+ Only on AIX V4.1 and later.						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						
* Supported on MQSeries for AIX version 2.2.1 or later.						
** Supported on MQSeries for HP version 2.2.1 or later.						

Code page conversion tables

<i>Table 32. Conversion support: GREEK</i>				
Target ▼	Source→	OS/2, AIX, HP-UX, GIS, Solaris, DEC-OVMS Tandem	OS/2, NT	MVS, OS/400
	CCSID	813	869	875
MVS	875	Y†	Y†	y
OS/400	875	Y	Y	y
OS/2	813	y	Y	Y
OS/2	869	Y	y	Y
AIX (ISO)	813	y	Y	Y
HP-UX (ISO)	813#	y	Y	Y
GIS (ISO)	813	y	Y	Y
NT	869	Y	y	Y
Solaris	813	y	Y	Y
SunOS				
SINIX, DC/OSx	813	y	Y	Y
DEC-OVMS	813	y	Y	Y
Tandem	813	y	Y	Y
Note:				
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.				
# Only the ISO codeset on HP-UX is supported. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.				

<i>Table 33. Conversion support: TURKISH</i>				
Target ▼	Source→	OS/2, NT	AIX, HP-UX, Solaris, DEC-OVMS Tandem	MVS, OS/400
	CCSID	857	920	1026
MVS	1026	Y†	Y†	<i>y</i>
OS/400	1026	Y	Y	<i>y</i>
OS/2	857	<i>y</i>		Y
AIX (ISO)	920	Y	<i>y</i>	Y
HP-UX (ISO)	920#	Y	<i>y</i>	Y
GIS				
NT	857	<i>y</i>	Y	Y
Solaris	920	Y	<i>y</i>	Y
SunOS				
SINIX, DC/OSx	920	Y	<i>y</i>	Y
DEC-OVMS	920	Y	<i>y</i>	Y
Tandem	920	Y	<i>y</i>	Y
Note:				
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.				
# Only the ISO codeset on HP-UX is supported. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.				

Code page conversion tables

<i>Table 34. Conversion support: HEBREW</i>					
Target ▼	Source→	MVS, OS/400	AIX	OS/2, NT	AIX, HP-UX, Solaris, DEC-OVMS Tandem
	CCSID	424	856	862	916
MVS	424	y	Y†	Y†	Y†
OS/400	424	y	Y#	Y	Y
OS/2	862	Y		y	
AIX (pc-A)	856	Y+	y	Y+	Y+
AIX (ISO)	916	Y+	Y+	Y+	y
HP-UX (ISO)	916§	Y	Y	Y	y
GIS					
NT	862	Y	Y	y	Y
Solaris	916	Y	Y	Y	y
SunOS					
SINIX, DC/OSx	916	Y	Y	Y	y
DEC-OVMS	916	Y	Y	Y	y
Tandem	916	Y	Y	Y	y
Note:					
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.					
# Only to/from CCSID 4952 (a variant of 856).					
+ Only on AIX V4.1 and later.					
§ Only the ISO codeset on HP-UX is supported. The HP-UX proprietary hebrew8 codeset has no registered CCSID and is not supported.					

<i>Table 35. Conversion support: ARABIC</i>					
Target ▼	Source→	MVS, OS/400	OS/2, NT	AIX	AIX, HP-UX, Solaris, DEC-OVMS Tandem
	CCSID	420	864	1046	1089
MVS	420	y	Y†	Y†	Y†
OS/400	420	y	Y	Y	
OS/2	864	Y	y		
AIX (pc-A)	1046	Y#	Y#	y	Y#
AIX (ISO)	1089	Y#	Y#	Y#	y
HP-UX (ISO)	1089§	Y	Y	Y	y
GIS					
NT	864	Y	y	Y	Y
Solaris	1089	Y	Y	Y	y
SunOS					
SINIX, DC/OSx	1089	Y	Y	Y	y
DEC-OVMS	1089	Y	Y	Y	y
Tandem	1089	Y	Y	Y	y
Note:					
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.					
# Only on AIX V4.1 and later.					
§ Only the ISO codeset on HP-UX is supported. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.					

Code page conversion tables

<i>Table 36. Conversion support: JAPANESE LATIN SBCS</i>					
Target ↓ ▼	Source→	OS/2, AIX	OS/2	MVS, OS/400	AIX
	CCSID	932	942	1027	5050 33722
MVS	1027			y	
OS/400	1027		Y	y	
OS/2	932	y	Y		
OS/2	942	Y	y		
AIX (pc-A)	932	y		Y	Y
AIX (euc)	5050 33722*	Y		Y	y
HP-UX					
GIS					
NT	932	y	Y	Y	
Solaris					
SunOS					
SINIX, DC/OSx					
DEC-OVMS					
Tandem					

Note: On AIX conversion from mixed DBCS to SBCS (OS/400 and MVS) will only convert the SBCS subset.

* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V3.2.5 MQSeries codes this code page as CCSID 5050 for compatibility with OS/400. On AIX V4.1 the CCSID reported by the operating system is 33722.

<i>Table 37. Conversion support: JAPANESE KATAKANA SBCS</i>					
Target ↓ ▼	Source→	MVS, OS/400	OS/2, HP-UX	AIX	AIX
	CCSID	290	897	932	5050 33722
MVS	290	y	Y		
OS/400	290	y	Y		
OS/2	897	Y	y		
AIX (pc-A)	932	Y		y	Y
AIX (euc)	5050 33722*	Y		Y	y
HP-UX (kana8)	897	Y	y		
GIS					
NT	932	Y	Y	y	
Solaris					
SunOS					
SINIX, DC/OSx					
DEC-OVMS					
Tandem					
<p>Note: On AIX conversion from mixed DBCS to SBCS (OS/400 and MVS) will only convert the SBCS subset.</p> <p>* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V3.2.5 MQSeries codes this code page as CCSID 5050 for compatibility with OS/400. On AIX V4.1 the CCSID reported by the operating system is 33722.</p>					

Code page conversion tables

<i>Table 38. Conversion support: JAPANESE KANJI / LATIN MIXED</i>						
Target ▼	Source→	OS/2, AIX, HP-UX, DEC-OVMS Tandem	OS/2	HP-UX, DEC-OVMS Tandem	MVS, OS/400	AIX
	CCSID	932	942	954	5035	5050 33722
MVS	5035#	Y†	Y†		y	
OS/400	5035#	Y	Y		y	Y
OS/2	932	y	Y		Y	
OS/2	942	Y	y		Y	
AIX (pc-A)	932	y		Y	Y	Y
AIX (ISO)	5050 33722*	Y		y	Y	y
HP-UX (euc)	954	Y		y	Y**	y
HP-UX (-15§)	932	y		Y	Y**	Y
GIS						
NT	932	y	Y		Y	
Solaris						
SunOS						
SINIX, DC/OSx						
DEC-OVMS	932	y		Y	Y	Y
DEC-OVMS	954	Y		y	Y	y
Tandem	932	y		Y	Y	Y
Tandem	954	Y		y	Y	y

Note:

- † Supported on MQSeries for MVS/ESA version 1.1.4 or later.
- * 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V3.2.5 MQSeries codes this code page as CCSID 5050 for compatibility with OS/400. On AIX V4.1 the CCSID reported by the operating system is 33722.
- # 5035 is a CCSID related to code page 939.
- § Defined by HP-UX as japan15 and SJIS. Note that about 74 DBCS characters have different representations in japan15 and 932 so may not be converted correctly if the conversion is performed on a non-HP-UX system.
- ** Supported on HP-UX V10 or later.

<i>Table 39. Conversion support: JAPANESE KANJI / KATAKANA MIXED</i>						
Target ▼	Source→	OS/2, AIX, HP-UX, DEC-OVMS Tandem	OS/2	HP-UX, DEC-OVMS Tandem	MVS, OS/400	AIX, Solaris
	CCSID	932	942	954	5026	5050 33722
MVS	5026#	Y†	Y†		y	
OS/400	5026#	Y	Y		y	Y
OS/2	932	y	Y		Y	
OS/2	942	Y	y		Y	
AIX (pc-A)	932	y		Y	Y	Y
AIX (euc)	5050 33722*	Y		y	Y	y
HP-UX (euc)	954	Y		y	Y**	y
HP-UX (-15§)	932	y		Y	Y	Y
GIS						
NT	932	y	Y		Y	
Solaris	5050	Y		y	Y	y
SunOS						
SINIX, DC/OSx						
DEC-OVMS (sjis)	932	y		Y	Y	Y
DEC-OVMS (euc)	954	Y		y	Y	y
Tandem (sjis)	932	y		Y	Y	Y
Tandem (euc)	954	Y		y	Y	y

Note:

- † Supported on MQSeries for MVS/ESA version 1.1.4 or later.
- * 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V3.2.5 MQSeries codes this code page as CCSID 5050 for compatibility with OS/400. On AIX V4.1 the CCSID reported by the operating system is 33722.
- # 5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported to the user on OS/400 when the Japanese Katakana (DBCS) feature is selected.
- § Defined by HP-UX as japan15 and SJIS. Note that about 74 DBCS characters have different representations in japan15 and 932 so may not be converted correctly if the conversion is performed on a non-HP-UX system.
- ** Supported on HP-UX V10 or later.

Code page conversion tables

<i>Table 40. Conversion support: KOREAN</i>				
Target ↓ ▼	Source→	MVS, OS/400	OS/2, NT	AIX, HP-UX, DEC-OVMS Tandem
	CCSID	933	949	970
MVS	933	y	Y†	
OS/400	933	y	Y	Y
OS/2	949	Y	y	
AIX (euc)	970	Y		y
HP-UX (-15)	949§	Y	y	
HP-UX (euc)	970§	Y		y
GIS				
NT	949	Y	y	
Solaris				
SunOS				
SINIX, DC/OSx				
DEC-OVMS	970	Y	Y	y
Tandem	970	Y	Y	y
Note:				
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.				
§ On HP-UX9 949 is used, but on HP-UX10 970 is used.				

<i>Table 41. Conversion support: SIMPLIFIED CHINESE</i>				
Target ▼	Source→	MVS, OS/400	OS/2, HP-UX, NT	AIX, DEC-OVMS Tandem
	CCSID	935	1381	1383
MVS	935	y	Y†	
OS/400	935	y	Y	Y+
OS/2	1381	Y	y	
AIX (euc)	1383*	Y*	Y*	y
HP-UX (-15)	1381§	Y**	y	
GIS				
NT	1381##	Y	Y	
Solaris				
SunOS				
SINIX, DC/OSx				
DEC-OVMS	1383	Y	Y	y
Tandem	1383	Y	Y	y
Note:				
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.				
+ Supported on OS/400 V3R6 or later.				
* Supported on country AIX version only.				
§ Is called prc15 and hp15CN on HP-UX.				
** Supported on HP-UX V10 or later.				
##				
NT uses the code page number 936, but this is best represented by the CCSID of 1381.				

Code page conversion tables

<i>Table 42. Conversion support: TRADITIONAL CHINESE</i>						
Target ▼	Source→	MVS, OS/400	OS/2, HP-UX	OS/2	OS/2, AIX, HP-UX, NT, DEC-OVMS Tandem	AIX, HP-UX, DEC-OVMS Tandem
	CCSID	937	938	948	950	964
MVS	937	y	Y†	Y†	Y†	
OS/400	937	y	Y	Y	Y	Y
OS/2 (PS/55)	938	Y	y			
OS/2 (PS/55)	948	Y		y		
OS/2 (big5)	950	Y			y	
AIX (euc)	964	Y	Y		Y	y
AIX (big5)	950	Y	Y		y	Y
HP-UX (-15§)	938	Y	y		Y	Y**
HP-UX (big5)	950	Y**	Y		y	Y**
HP-UX (eucTW)	964	Y**	Y**		Y**	y
GIS						
NT	950	Y	Y	Y	y	
Solaris						
SunOS						
SINIX, DC/OSx						
DEC-OVMS (euc)	964	Y	Y	Y	Y	y
DEC-OVMS (big5)	950	Y	Y	Y	y	Y
Tandem (euc)	964	Y	Y	Y	Y	y
Tandem (big5)	950	Y	Y	Y	y	Y
Note:						
† Supported on MQSeries for MVS/ESA version 1.1.4 or later.						
§ Is called roc15 and eucTW on HP-UX.						
** Supported on HP-UX V10 or later.						

<i>Table 43. Codeset names and CCSIDs</i>	
Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
ISO 8859-5	915
ISO 8859-6	1089
ISO 8859-7	813
ISO 8859-8	916
ISO 8859-9	920
eucJP	954 5050 33722
eucKR	970
eucTW	964
eucCN	1383

MVS single byte conversion support

Table 44 (Page 1 of 6). MVS V1.1.4 single byte CCSID conversion support.

CCSID	Converts to and from CCSIDS
37	256, 273, 275, 277, 278, 280, 284, 285, 290, 297, 367, 420, 423, 424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903-905, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1114, 1252, 1275
256	37, 273, 277, 278, 280, 284, 285, 290, 297, 367, 420, 423, 424, 437, 500, 819, 833, 836, 838, 850, 852, 857, 860-866, 869-871, 875, 880, 905, 1025-1027, 1251, 1252, 1275
259	437, 850-852, 855-857, 860-865, 869, 874, 899, 915, 1098, 1251
273	37, 256, 277, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855-857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
274	500, 1047
275	37, 500, 1047
277	37, 256, 273, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
278	37, 256, 273, 277, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
280	37, 256, 273, 277, 278, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
281	1047
282	500, 1047
284	37, 256, 273, 277, 278, 280, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
285	37, 256, 273, 277, 278, 280, 284, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
290	37, 256, 273, 277, 278, 280, 284, 285, 297, 367, 437, 500, 819, 833, 836, 850, 852, 855, 857, 860-865, 870, 871, 895-897, 1009, 1025-1027, 1040-1043, 1088
297	37, 256, 273, 277, 278, 280, 284, 285, 290, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1252, 1275
367	37, 256, 273, 277, 278, 280, 284, 290, 297, 500, 833, 836, 871, 875, 1009, 1026, 1027, 1041, 1088, 1115

Table 44 (Page 2 of 6). MVS V1.1.4 single byte CCSID conversion support.

CCSID	Converts to and from CCSIDS
420	37, 256, 424, 437, 500, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1256
423	37, 256, 273, 277, 278, 280, 284, 285, 297, 437, 500, 813, 819, 838, 850-852, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1253, 1280
424	37, 256, 420, 437, 500, 803, 819, 836, 850, 852, 856, 857, 860-865, 916, 1255
437	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-863, 865, 866, 869-871, 874, 875, 880, 897, 903, 905, 912, 915, 916, 920, 1025-1027, 1040-1043, 1051, 1097, 1098, 1252, 1275, 4946, 28709
500	37, 256, 273-275, 277, 278, 280, 282, 284, 285, 290, 297, 367, 420, 423, 424, 437, 813, 819, 833, 836, 838, 850-852, 855-857, 860-866, 869-871, 874, 875, 880, 891, 895, 897, 903-905, 912, 915, 916, 920, 1004, 1009-1021, 1023, 1025-1027, 1040-1043, 1046, 1047, 1051, 1088, 1089, 1097, 1100-1107, 1114, 1115, 1250-1256, 1275
803	424, 856, 862, 916
813	37, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 819, 838, 850, 852, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1253, 1280
819	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 833, 836, 838, 850, 852, 857, 860, 861, 863, 865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1047, 1051, 1097, 1098, 1114, 1252, 1275
833	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 367, 437, 500, 819, 836, 850, 852, 855, 857, 860-865, 870, 871, 891, 1009, 1025-1027, 1040-1043, 1088
836	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 367, 424, 437, 500, 819, 833, 850, 852, 855, 857, 870, 871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1115
838	37, 256, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 850, 852, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043
850	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 836, 838, 852, 855-857, 860-866, 869-871, 874, 875, 880, 897, 903, 905, 912, 915, 916, 920, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1098, 1100, 1114, 1252, 1275, 4953
851	259, 423, 500, 875
852	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 836, 838, 850, 855, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1250, 1282, 28709
855	37, 259, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 833, 836, 850, 852, 857, 866, 870, 871, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1251, 1283
856	259, 273, 424, 500, 803, 850, 862, 916, 1255

Code page conversion tables

<i>Table 44 (Page 3 of 6). MVS V1.1.4 single byte CCSID conversion support.</i>	
CCSID	Converts to and from CCSIDS
857	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1254, 1281, 28709
860	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 861, 863, 865, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1097, 28709
861	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 860, 863, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1097, 28709
862	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 803, 833, 838, 850, 856, 870, 871, 875, 880, 905, 916, 1025-1027, 1097, 1255, 28709
863	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 860, 861, 865, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1051, 1097, 1252, 1275, 28709
864	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 500, 833, 838, 850, 870, 871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1256, 28709
865	37, 256, 259, 273, 277, 278, 280, 284, 285, 290, 297, 420, 423, 424, 437, 500, 819, 833, 838, 850, 860, 863, 870, 871, 875, 880, 905, 1025-1027, 1097, 28709
866	256, 437, 500, 850, 855, 870, 880, 915, 1025, 1251, 1283
868	918
869	37, 256, 259, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860, 861, 863, 870, 871, 874, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1253, 1254, 1280
870	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869, 871, 874, 875, 880, 897, 903, 912, 915, 916, 920, 1009, 1025-1027, 1040-1043, 1088, 1250, 1282
871	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869, 870, 874, 875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1252, 1275
874	37, 259, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860, 861, 863, 869-871, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043
875	37, 256, 273, 277, 278, 280, 284, 285, 297, 367, 423, 437, 500, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1253, 1280
880	37, 256, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 855, 857, 860-866, 869-871, 874, 875, 897, 903, 912, 915, 916, 920, 1009, 1025-1027, 1041-1043, 1251, 1283
891	500, 833, 1088

Table 44 (Page 4 of 6). MVS V1.1.4 single byte CCSID conversion support.

CCSID	Converts to and from CCSIDS
895	290, 500, 1027, 1041
896	290, 1027, 1041
897	37, 273, 277, 278, 280, 284, 285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860, 861, 863, 869-871, 874, 875, 880, 903, 912, 916, 920, 1025-1027, 1041-1043
899	259
903	37, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 912, 916, 920, 1025-1027, 1041-1043, 1115
904	37, 500, 1114
905	37, 256, 437, 500, 850, 852, 857, 860-865, 920, 1026, 1254, 1281
912	37, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 855, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 916, 920, 1025-1027, 1041-1043, 1250, 1282
915	259, 437, 500, 850, 855, 866, 870, 880, 1025, 1251, 1283
916	37, 273, 277, 278, 280, 284, 285, 297, 423, 424, 437, 500, 803, 813, 819, 838, 850, 852, 856, 857, 860-863, 869-871, 874, 875, 880, 897, 903, 912, 920, 1025-1027, 1041-1043, 1255
918	864, 868
920	37, 273, 277, 278, 280, 284, 285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 1025, 1026, 1254, 1281
932	942
942	932
1004	500
1008	420, 864
1009	37, 273, 277, 278, 280, 284, 290, 297, 367, 423, 500, 833, 836, 870, 871, 875, 880, 1025, 1026
1010	500
1011	500
1012	500
1013	500
1014	500
1015	500
1016	500
1017	500
1018	500
1019	500
1020	500
1021	500
1023	500

Code page conversion tables

<i>Table 44 (Page 5 of 6). MVS V1.1.4 single byte CCSID conversion support.</i>	
CCSID	Converts to and from CCSIDS
1025	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869-871, 874, 875, 880, 897, 903, 912, 915, 916, 920, 1009, 1026, 1027, 1040-1043, 1051, 1088, 1251, 1283
1026	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 897, 903, 905, 912, 916, 920, 1009, 1025, 1027, 1040-1043, 1047, 1088, 1254, 1281
1027	37, 256, 273, 277, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874, 875, 880, 895-897, 903, 912, 916, 1025, 1026, 1040-1043, 1047, 1088
1040	37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 833, 836, 850, 852, 855, 857, 870, 871, 1025-1027, 1041-1043, 1088
1041	37, 273, 277, 278, 280, 284, 285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860, 861, 863, 869-871, 874, 875, 880, 895-897, 903, 912, 916, 1025-1027, 1040, 1042, 1043, 1088
1042	37, 273, 277, 278, 280, 284, 285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1043, 1088
1043	37, 273, 277, 278, 280, 284, 285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860, 861, 863, 869-871, 874, 875, 880, 897, 903, 912, 916, 1025-1027, 1040-1042, 1088, 1114
1046	420, 500, 864, 1089, 1256
1047	37, 273-275, 277, 278, 280-282, 284, 285, 297, 500, 819, 850, 871, 875, 1026, 1027
1051	37, 273, 277, 278, 280, 284, 285, 297, 437, 500, 819, 850, 863, 871, 1025, 1097, 1252, 1275
1088	37, 273, 277, 278, 280, 284, 285, 290, 297, 367, 500, 833, 836, 850, 852, 855, 857, 870, 871, 875, 891, 1025-1027, 1040-1043
1089	420, 500, 864, 1046, 1256
1097	37, 437, 500, 819, 850, 852, 857, 860-865, 1051, 1098
1098	259, 420, 437, 819, 850, 1097
1100	37, 273, 277, 278, 280, 284, 285, 297, 500, 850
1101	500
1102	500
1103	500
1104	500
1105	500
1106	500
1107	500
1114	37, 500, 819, 850, 904, 1043
1115	367, 500, 836, 903
1250	500, 852, 870, 912, 1282

Table 44 (Page 6 of 6). MVS V1.1.4 single byte CCSID conversion support.

CCSID	Converts to and from CCSIDS
1251	256, 259, 500, 855, 866, 880, 915, 1025, 1283
1252	37, 256, 273, 277, 278, 280, 284, 285, 297, 437, 500, 819, 850, 863, 871, 1051, 1275
1253	423, 500, 813, 869, 875, 1280
1254	500, 857, 869, 905, 920, 1026, 1281
1255	424, 500, 856, 862, 916
1256	420, 500, 864, 1046, 1089
1275	37, 256, 273, 277, 278, 280, 284, 285, 297, 437, 500, 819, 850, 863, 871, 1051, 1252
1280	423, 813, 869, 875, 1253
1281	857, 905, 920, 1026, 1254
1282	852, 870, 912, 1250
1283	855, 866, 880, 915, 1025, 1251
4946	437
4953	850
28709	437, 852, 857, 860-865

Code page conversion tables

Appendix H. Stopping and removing queue managers manually

If the normal methods for stopping and removing queue managers fail, you can resort to the more drastic methods described here.

Stopping queue managers manually

Attention Both the MQ interprocess communication monitor (ipc) commands and the VMS STOP command are powerful functions. As a system manager you should use extreme care in using them.

In exceptional circumstances, if **endmqm** fails, use the following procedure to stop it manually:

1. Find the process Ids of the queue manager programs that are still running, by using the MQ ipc monitor. To do this, type `$ monmq active`.

Suppose that there are two active queue managers, "qm0" and "qm1". The following is an example of what might be displayed:

```

id   pid  ch  us  st  hb  sm  wp  to  lk  nq  we  sp  wk  ws  estat  prog
000 000f60 11  1  0  0  0  0  0  0  0  0  0  0  0  78f08054 / MONMQIPC.EXE
001 001039 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQZXMA0.EXE
002 00103b 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQHASM.X.EXE
003 00103c 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQZLLP0.EXE
010 001045 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm1/ AMQZXMA0.EXE
011 001046 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm1/ AMQHASM.X.EXE
012 000247 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm1/ AMQZLLP0.EXE
018 001082 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQZLAA0.EXE
Proc:117 Next:126 MsgW Val/Cnt:240/0 MsgR Val/Cnt:0/0 WakCnt:1 Evt:490 Region Ower Pidx:0 Tid:0

```

All the processes associated with a given queue manager have the name of that queue manager appear before the "/" under the prog column – the last column in the 'active' display.

2. End the queue manager by stopping its 'execution (processing) controller'. Use the ipc monitor kill command to do this as follows.

Enter:

- `monmq kill id`
- The `id` as shown under the `id` column for the program named `AMQZXMA0`, for the queue manager you want to stop.

For queue manager `qm1`, in the example shown, you enter

```
monmq kill 10
```

to stop the execution controller for queue manager `qm1`. Subsequently, the other sub-processes will stop.

3. Continue to display all processes until you are certain that all processes for the desired queue manager have stopped. Use the ipc monitor again to find these active processes, by typing `$ monmq active`

When queue manager, `qm1`, has gone the following will display:

```

id   pid  ch  us  st  hb  sm  wp  to  lk  nq  we  sp  wk  ws  estat  prog
000 000f60 11  1  0  0  0  0  0  0  0  0  0  0  0  78f08054 / MONMQIPC.EXE
001 001039 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQZXMA0.EXE
002 00103b 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQHASM.X.EXE
003 00103c 11  1  0  1  1  0  0  0  1  0  0  0  0  0000 qm0/ AMQZLLP0.EXE

```

Removing queue managers

The example shows that only queue manager `qm0` is still active.

In the extremely rare case, where not all of the queue manager processes stop for the desired queue manager, so that they still appear in the display, you can use the VMS stop command, as follows:

```
$ stop proc/id=aaaa
```

where `aaaa` is the pid found under the pid column in the `monmq` active display.

Note: Use the pid in column two, not the id in column one.

End the processes in the following order:

1. `amqhasmx` – logger
2. `amqharmx` – log formatter, used only if the queue manager has linear logging selected
3. `amqz11p0` – checkpoint processor
4. `amqz1aa0` – queue manager agents
5. `amqzma0` – processing controller

Note: Manual ending of the queue manager may result in FFSTs being generated, and the production of FDC files in `MQS_ROOT:[MQM.ERRORS]`. This should not be regarded as a defect in the queue manager.

The queue manager should restart normally, even if it was ended by using the preceding method.

If you want to delete the queue manager after stopping it manually, use the `dlmqm` command as normal. If, for some reason, this command fails to delete the queue manager, the manual process detailed in “Removing queue managers manually” can be used.

Removing queue managers manually

You should note that manual removal of a queue manager is potentially very disruptive, particularly if multiple queue managers are being used on a single system. 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. Look in `MQS_ROOT:[MQM]MQS.INI` for the stanza relating to the queue manager that you want to delete.

Use the prefix and directory fields to identify the location of the queue manager directory structure, by concatenating the prefix and directory fields as follows:

```
<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 MQS_ROOT:[MQM]MQS.INI configuration file.
7. If the queue manager being deleted is also the default queue manager, remove the DefaultQueueManager stanza from the MQS_ROOT:[MQM]MQS.INI configuration file.
8. Stop all queue managers.
9. Either, delete all global sections by using `monmq delete`, or IPL the machine.

Removing queue managers

Appendix I. Building your application on Digital OpenVMS

Note

The information in this appendix will be inserted into the *MQSeries Application Programming Guide* the next time that this book is refreshed.

This appendix describes the additional tasks, and the changes to the standard tasks, you must perform when building MQSeries for Digital OpenVMS applications to run under Digital OpenVMS. C and COBOL are supported.

In addition to coding the MQI calls in your source code, you must add the appropriate include files. You should make yourself familiar with the contents of these files—their names, and a brief description of their contents are given in the following section.

MQSeries for Digital OpenVMS data definition files

MQSeries for Digital OpenVMS provides data definition files to assist you with the writing of your applications. These data definition files are also known as:

Language	Data definitions
C	Include files or header files
COBOL	Copy files

See the *Application Programming Guide* for the compilers that are supported and suitable for use with these data definition files.

The data definition files to assist with the writing of channel exits are described in the *Distributed Queuing Guide*.

The data definition files to assist with the writing of installable services exits are described in the *MQSeries Programmable System Management Guide*.

C language include files

The MQSeries for Digital OpenVMS C include files are listed in Table 45. They are installed in the MQS_INCLUDE directory.

File name	Contents
<cmqc.h>	Call prototypes, data types, structures, return codes, and constants
<cmqfc.h>	Definitions for programmable commands
<cmqxc.h>	Definitions for channel exits and data conversion exits
<cmqzc.h>	Definitions for installable services exits

Data types

All data types are defined by means of the typedef statement. For each data type, the corresponding pointer data type is also defined. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter “P” to denote a pointer; for example:

```
typedef MQLONG MQPOINTER PMQLONG; /* pointer to MQLONG */
typedef MQMD MQPOINTER PMQMD; /* pointer to MQMD */
```

Initial values for structures

The include file <cmqc.h> defines various macro variables that may be used to provide initial values for the structures when instances of those structures are declared. These macro variables have names of the form MQxxx_DEFAULT, where MQxxx represents the name of the structure. Use them like this:

```
MQMD MyMsgDesc = {MQMD_DEFAULT};
MQPMO MyPutOpts = {MQPMO_DEFAULT};
```

For some character fields, the MQI defines particular values that are valid (for example, for the *StrucId* fields or for the *Format* field in MQMD). For each of the valid values, *two* macro variables are provided:

- One macro variable defines the value as a string whose length, excluding the implied null, matches exactly the defined length of the field. For example, (the symbol *b* represents a blank character):

```
#define MQMD_STRUC_ID "MDbb"
#define MQFMT_STRING "MQSTRbbb"
```

Use this form with the *memcpy* and *memcmp* functions.

- The other macro variable defines the value as an array of char; the name of this macro variable is the name of the string form suffixed with “_ARRAY”. For example:

```
#define MQMD_STRUC_ID_ARRAY 'M','D','b','b'
#define MQFMT_STRING_ARRAY 'M','Q','S','T','R','b','b','b'
```

Use this form to initialize the field when an instance of the structure is declared with values different from those provided by the MQMD_DEFAULT macro variable.

COBOL copy files

For COBOL, MQSeries for Digital OpenVMS provides separate copy files containing the named constants, and two copy files for each of the structures. There are two copy files for each structure because each is provided both with and without initial values:

- In the WORKING-STORAGE SECTION of a COBOL program, use the files that initialize the structure fields to default values. These structures are defined in the copy files that have names suffixed with the letter “V” (values).
- In the LINKAGE SECTION of a COBOL program, use the structures without initial values. These structures are defined in copy files that have names suffixed with the letter “L” (linkage).

The MQSeries for Digital OpenVMS COBOL copy files are listed in Table 46. They are installed in the MQS_INCLUDE directory.

<i>Table 46. COBOL copy files for MQSeries for Digital OpenVMS</i>		
File name (with initial values)	File name (without initial values)	Contents
CMQDLHV.LIB	CMQDLHL.LIB	Dead-letter (undelivered-message) header structure (MQDLH)
CMQGMOV.LIB	CMQGMOL.LIB	Get-message options structure (MQGMO)
CMQMDV.LIB	CMQMDL.LIB	Message descriptor structure (MQMD)
CMQODV.LIB	CMQODL.LIB	Object descriptor structure (MQOD)
CMQPMOV.LIB	CMQPMOL.LIB	Put-message options structure (MQPMO)
CMQTMV.LIB	CMQTMOL.LIB	Trigger-message structure (character format) (MQTMC)
CMQTMV.LIB	CMQTML.LIB	Trigger-message structure (MQTM)
CMQV.LIB	not applicable	Named constants for the MQI
CMQXQHV.LIB	CMQXQHL.LIB	Transmission-queue header structure (MQXQH)

Include in your program only those files you need. Do this with one or more COPY statements after a level-01 declaration (see Figure 17 for an example). This means you can include multiple versions of the structures in a program if necessary. However, note that CMQV is a large file.

```
01-MQM-MESSAGE-DESCRIPTOR.
  COPY "MQS_INCLUDE:CMQMDV"
```

Figure 17. Example of COBOL code for including the CMQMDV copy file

Each structure declaration begins with a level-10 item; this means you can declare several instances of the structure by coding the level-01 declaration followed by a COPY statement to copy in the remainder of the structure declaration. To refer to the appropriate instance, use the IN keyword, as shown in Figure 18.

```
* Declare two instances of MQMD
01 MY-CMQMD.
  COPY "MQS_INCLUDE:CMQMDV"
01 MY-OTHER-CMQMD.
  COPY "MQS_INCLUDE:CMQMDV"
*
* Set MSGTYPE field in MY-OTHER-CMQMD
  MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

Figure 18. Example of COBOL code for including two instances of CMQMDV

The structures should be aligned on 4-byte boundaries. If you use the COPY statement to include a structure following an item that is not the level-01 item, try to ensure that the structure is a multiple of 4-bytes from the start of the level-01 item. If you do not do this, you may get a reduction in the performance of your application.

In the descriptions of the fields of structures in the *MQSeries Application Programming Reference* fields are shown without a prefix. In COBOL programs you must prefix the field names with the name of the structure followed by a hyphen; this is shown in the COBOL declarations in the *MQSeries Application Programming Reference*. The fields in the structure copy files are prefixed this way.

The field names in the declarations in the *MQSeries Application Programming Reference* and in the structure copy files are in uppercase. You can use mixed case or lowercase instead. For example, the field *StrucId* of the MQGMO structure is shown as MQGMO-STRUCID in the COBOL declaration and in the copy file.

The V-suffix structures are declared with initial values for all of the fields, so you need to set only those fields where the value required is different from the initial value.

Named constants

In this book, the names of constants are shown containing the underscore character (_) as part of the name. In COBOL, you must use the hyphen character (-) in place of the underscore.

The copy file CMQV contains declarations of the named constants as level-10 items. To use the constants, declare the level-01 item explicitly, then use the COPY statement to copy in the declarations of the constants (see Figure 19).

```
WORKING-STORAGE SECTION.  
01 MQM-CONSTANTS.  
   COPY "MQS_INCLUDE:CMQV"
```

Figure 19. Example of COBOL code for declaring constants

However, this method causes the constants to occupy storage in the program even if they are not referred to. If the constants are included in many separate programs within the same run unit, multiple copies of the constants will exist—this may result in a significant amount of main storage being used. You can avoid this situation by adding the GLOBAL clause to the level-01 declaration (see Figure 20).

```
* Declare a global structure to hold the constants  
01 MQM-CONSTANTS GLOBAL.  
   COPY "MQS_INCLUDE:CMQV"
```

Figure 20. Example of COBOL code using the GLOBAL clause

This causes storage to be allocated for only *one* set of constants within the run unit; the constants, however, can be referred to by *any* program within the run unit, not just the program that contains the level-01 declaration.

Preparing C programs

This section explains the compiler and libraries you need to prepare your C programs.

C Compiler version

You must use the DEC C compiler. To invoke the compiler, enter:

```
$ CC/DECC
```

This is the default.

C compiler flags

The include files for MQSeries for Digital OpenVMS are located in the MQS_INCLUDE directory. The following is an example of how to build the sample program AMQSPUT0:

```
$ CC/INCLUDE_DIRECTORY=MQS_INCLUDE AMQSPUT0
$ LINK AMQSPUT0.OBJ,SYS$INPUT/OPTIONS
SYS$SHARE: MQM/SHAREABLE
Ctrl + Z
```

Linking libraries

You need to link your programs with the appropriate library provided by MQSeries. The libraries are found in SYS\$SHARE.

You must link to one or more of the following libraries:

Library file	Program/exit type
<i>mqm.exe</i>	C
<i>mqic.exe</i>	MQSeries client (C only)
<i>mqmzf.exe</i>	installable service

Preparing COBOL programs

This section explains the compiler and libraries you need to prepare your COBOL programs.

COBOL compiler flags

You must compile the programs in ANSI mode using the /ANSI switch to the DEC COBOL compiler. The following is an example of how to build the sample program AMQ0PUT0:

```
$ COBOL/ANSI AMQ0PUT0.COB
$ LINK AMQ0PUT0.OBJ,SYS$INPUT/OPTIONS
SYS$SHARE: MQMCB/SHAREABLE
Ctrl + Z
```

Linking libraries

You need to link your program with one of the following:

MQMCB.EXE	COBOL
MQICB.EXE	COBOL MQSeries client

User exits

The user exit is a dynamically loaded shareable image whose name is taken from the format of the message. The object's name must be in uppercase, for example MYFORMAT. The shareable image must be placed in sys\$share or a location defined by a logical name at executive level for it to be loaded.

User exits must be installed as known images.

In the example, MQSTART is the initialization routine entry point for the MYFORMAT shareable image. The names of the routines which are called by the data-conversion exit must be made universal.

```
$ CC /INCLUDE_DIRECTORY=MQS_INCLUDE AMQSVFCX.C
$ LINK /SHARE=SYS$SHARE:MYFORMAT AMQSVFCX.OBJ,MYFORMAT/OPTIONS
```

The contents of MYFORMAT.OPT vary depending on what platform you are working on:

On AXP:

```
SYS$SHARE:MQM/SHAREABLE
SYMBOL_VECTOR=(MQSTART=PROCEDURE)
```

On VAX:

```
SYS$SHARE:MQM/SHAREABLE
UNIVERSAL=MQSTART
```

If you are using threaded applications linked with the pthread library, you must also build a second copy of the data-conversion exit with the thread options and libraries:

```
$ CC /INCLUDE_DIRECTORY=MQS_INCLUDE AMQSVFCX.C
$ LINK /SHARE=SYS$SHARE:MYFORMAT AMQSVFCX.OBJ,MYFORMAT/OPTIONS
```

Again, the contents of MYFORMAT.OPT vary depending on what platform you are working on:

On AXP:

```
SYS$SHARE:MQM_R/SHAREABLE
SYS$SHARE:CMA$OPEN_RTL.EXE/SHAREABLE
SYMBOL_VECTOR=(MQSTART=PROCEDURE)
```

On VAX:

```
SYS$SHARE:MQM_R/SHAREABLE
SYS$SHARE:CMA$OPEN_RTL.EXE/SHAREABLE
UNIVERSAL=MQSTART
```

See the *Application Programming Guide* for more details on data-conversion exits.

Appendix J. Monitoring and controlling DQM on OS/2, Windows NT, UNIX systems and Digital OpenVMS

Note

The information in this appendix will be inserted into the *MQSeries Distributed Queuing Guide* the next time that this book is refreshed.

For DQM you need to create, monitor, and control the channels to remote queue managers. You can use the following types of commands to do this:

The MQSeries commands (MQSC)

You can use the MQSC as single commands in an MQSC session in OS/2, Windows NT, UNIX systems and Digital OpenVMS systems. To issue more complicated, or multiple commands, the MQSC can be built into a file that you then run from the command line. For full details see the *MQSeries Command Reference* manual.

This chapter gives some simple examples of using MQSC for distributed queuing.

Control commands

You can also issue *control commands* at the command line for some of these functions. Reference material for these commands is contained in the *MQSeries System Management Guide* for your platform.

Programmable command format commands

See the *MQSeries Programmable System Management* manual for information about using these commands.

Each queue manager has a DQM component for controlling interconnections to compatible remote queue managers.

For a list of the functions available to you when setting up and controlling message channels, using the two types of commands, see Table 47 on page 286.

The DQM channel control function

The channel control function provides the interface and function for administration and control of message channels between systems.

It consists of commands, programs, a synchronization queue and file, and a file for the channel definitions. The following is a brief description of the components:

- The channel definition file (CDF):
 - Is indexed on channel name
 - Holds channel definitions
- The channel commands are a subset of the MQSeries Commands.
- You use MQSC and the control commands to:
 - Create, copy, display, change, and delete channel definitions

Functions available

- Start and stop channels, ping, reset channel sequence numbers, and resolve in-doubt messages when links cannot be re-established
- Display status information about channels
- A synchronization queue and file hold sequence numbers and *logical unit of work (LUW)* identifiers. They are used for channel synchronization purposes.

Functions available

Table 47 is the full list of MQSeries functions that you may need when setting up, and controlling channels. The channel functions are explained in this chapter.

For more details of the control commands that you issue at the command line, see the *MQSeries System Management Guide* for your platform.

The MQSC commands are fully described in the *MQSeries Command Reference*.

<i>Table 47 (Page 1 of 2). Functions available in MQSeries for OS/2, Windows NT, UNIX systems and Digital OpenVMS systems</i>		
Function	Control commands	MQSC
<i>Queue manager functions</i>		
Change queue manager		ALTER QMGR
Create queue manager	crtmqm	
Delete queue manager	dltmqm	
Display queue manager		DISPLAY QMGR
End queue manager	endmqm	
Ping queue manager		PING QMGR
Start queue manager	strmqm	
<i>Command server functions</i>		
Display command server	dspmqcsv	
End command server	endmqcsv	
Start command server	strmqcsv	
<i>Queue functions</i>		
Change queue		ALTER QALIAS ALTER QLOCAL ALTER QMODEL ALTER QREMOTE
Clear queue		CLEAR QLOCAL
Create queue		DEFINE QALIAS DEFINE QLOCAL DEFINE QMODEL DEFINE QREMOTE
Delete queue		DELETE QALIAS DELETE QLOCAL DELETE QMODEL DELETE QREMOTE

<i>Table 47 (Page 2 of 2). Functions available in MQSeries for OS/2, Windows NT, UNIX systems and Digital OpenVMS systems</i>		
Function	Control commands	MQSC
Display queue		DISPLAY QUEUE
<i>Process functions</i>		
Change process		ALTER PROCESS
Create process		DEFINE PROCESS
Delete process		DELETE PROCESS
Display process		DISPLAY PROCESS
<i>Channel functions</i>		
Change channel		ALTER CHANNEL
Create channel		DEFINE CHANNEL
Delete channel		DELETE CHANNEL
Display channel		DISPLAY CHANNEL
Display channel status		DISPLAY CHSTATUS
End channel		STOP CHANNEL
Ping channel		PING CHANNEL
Reset channel		RESET CHANNEL
Resolve channel		RESOLVE CHANNEL
Run channel	runmqchl	START CHANNEL
Run channel initiator	runmqchi	START CHINIT
Run listener (OS/2 and Windows NT only)	runmqlsr	START LISTENER
<i>Other functions</i>		
Display the files used by MQSeries objects	dspmqfls	
Record MQSeries object image	rcdmqimg	
Recreate MQSeries object	rcrmqobj	
Run MQSC	runmqsc	
Create MQSeries conversion exit	crtmqcvx	
Run trigger monitor	runmqtrm	
Display transaction	dspmqtrn	
Resolve transaction	rsvmqtrn	

Getting started

Use the MQSeries commands (MQSC) to:

1. Define message channels and associated objects
2. Monitor and control message channels

The objects you need to define are:

- Transmission queues
- Remote queue definitions
- Queue manager alias definitions
- Reply-to queue alias definitions
- Reply-to local queues
- Processes for triggering (MCAs)
- Message channel definitions

Channels must be completely defined, and their associated objects must exist and be available for use, before a channel can be started. This chapter shows you how to do this.

In addition, the particular communication link for each channel must be defined and available before a channel can be run. For a description of how LU 6.2, TCP/IP, and DECnet links are defined, see the particular communication guide for your installation.

Creating objects

Use MQSC to create the queue and alias objects: transmission queues, remote queue definitions, queue manager alias definitions, reply-to queue alias definitions, and reply-to local queues.

Also create the definitions of processes for triggering (MCAs) in a similar way.

For an example showing how to create all the required objects, in one particular situation, see the Distributed Queuing Guide.

Creating a channel

To create a new channel you have to create **two** channel definitions, one at each end of the connection. You create the first channel definition at the first queue manager. Then you create the second channel definition at the second queue manager, on the other end of the link.

Both ends must be defined using the **same** channel name.

The two ends must have **compatible** channel types, for example: Sender and Receiver.

To create a channel definition for one end of the link use MQSC DEFINE CHANNEL. Include the name of the channel, the channel type for this end of the connection, a description (if required), the name of the transmission queue, and the transmission protocol. Also include any other attributes that you want to be different from the system default values for the required channel type, using the information you have gathered previously.

You are provided with help in deciding on the values of the channel attributes. See the *MQSeries Distributed Queuing Guide*.

Note: You are very strongly recommended to name all the channels in your network uniquely. Including the source and target queue manager names in the channel name is a good way to do this.

Create channel example

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) +
DESCR('Sender channel to QM2') +
CONNNAME(QM2) TRPTYPE(TCP) XMITQ(QM2) CONVERT(YES)
```

In all the examples of MQSC the command is shown as it would appear in a file of commands, and as it would be typed in OS/2, Windows NT, UNIX systems, or Digital OpenVMS. The two methods look identical, except that to issue a command in OS/2, Windows NT, or UNIX systems you must first start an MQSC session. Type `runmqsc`, for the default queue manager, or `runmqsc QMNAME` where QMNAME is the name of the required queue manager. Then type any number of commands, as shown in the examples.

There is an 80-character limit to the line length. Use a concatenation character as shown to continue over more than one line. On OS/2 Windows NT, or Digital OpenVMS use Ctrl-z to end the input at the command line. On UNIX systems, use Ctrl-d to end the input at the command line.

Displaying a channel

Use MQSC `DISPLAY CHANNEL`, specifying the channel name, the channel type (optional) and the attributes you want to see, or specifying that all attributes are displayed.

The attributes are described in the *MQSeries Distributed Queuing Guide*.

Display channel examples

```
DISPLAY CHANNEL(QM1.TO.QM2) TRPTYPE,CONVERT
```

```
DISPLAY CHANNEL(QM1.TO.*) TRPTYPE,CONVERT
```

```
DISPLAY CHANNEL(*) TRPTYPE,CONVERT
```

```
DISPLAY CHANNEL(QM1.TO.QMR34) ALL
```

Display channel status

Use MQSC DISPLAY CHSTATUS specifying the channel name and whether you want the current status of channels or the status of saved information.

Display channel status examples

```
DISPLAY CHSTATUS(QM1.TO.*) SAVED
```

```
DISPLAY CHSTATUS(*) CURRENT
```

Starting a channel

For applications to be able to exchange messages you must start a listener program for inbound connections (create a listener attachment in UNIX systems). In OS/2 and Windows NT you can use the **runmqlsr** command to start channels as threads.

For outbound connections you must start the channel. Use MQSC **START CHANNEL**, specifying the channel name, to start the channel as a process or a thread, based on the MCATYPE parameter. Alternatively you can start channels as processes using the control command **runmqchl**.

Start channel examples

```
START CHANNEL(QM1.TO.QM2)
```

```
runmqchl -c QM1.TO.QM2 -m QM1
```

Renaming a channel

To rename a message channel, use MQSC to carry out the following steps:

1. Use STOP CHANNEL to stop the channel.
2. Use DEFINE CHANNEL to create a duplicate channel definition with the new name.
3. Use DISPLAY CHANNEL to check that it has been created correctly.
4. Use DELETE CHANNEL to delete the original channel definition.

If you decide to rename a message channel, remember that a channel has **two** channel definitions, one at each end. Make sure you rename the channel at both ends at the same time.

Channel attributes and channel types

The channel attributes that are required for each type of channel are shown here in Table 48.

The channel attributes are described in detail in the *MQSeries Distributed Queuing Guide*.

Table 48 (Page 1 of 2). Channel attributes for the channel types in OS/2, Windows NT, UNIX systems and Digital OpenVMS

Attribute field	Sender	Server	Receiver	Requester	Client-connection	Server-connection
Batch size	√	√	√	√		
Channel name	√	√	√	√	√	√
Channel type	√	√	√	√	√	√
Connection name	√	O		O	√	
Convert message	√	√				
Disconnect interval	√	√				
Long retry wait interval	√	√				
Long retry count	√	√				
LU 6.2 Transaction program name	O	O		O	O	
LU 6.2 Transmission program name	O	O		O	O	
Maximum message length	√	√	√	√		
Message channel agent type	√	√		√	√	
Message channel agent user	O	O	O	O	O	O
Message exit name	O	O	O	O		
Message exit user data	O	O	O	O		
Message-retry exit name			O	O		
Message-retry exit user data			O	O		
Message retry count			O	O		
Message retry interval			O	O		
Mode name	O	O		O	O	
Password	O	O		O	O	
Queue manager name					√	
PUT authority			√	√		
Receive exit	O	O	O	O	O	O
Receive exit user data	O	O	O	O	O	O
Security exit	O	O	O	O	O	O
Security exit user data	O	O	O	O	O	O
Send exit	O	O	O	O	O	O
Send exit user data	O	O	O	O	O	O
Sequence number wrap	√	√	√	√		
Short retry wait interval	√	√				
Short retry count	√	√				

Channel functions

Table 48 (Page 2 of 2). Channel attributes for the channel types in OS/2, Windows NT, UNIX systems and Digital OpenVMS

Attribute field	Sender	Server	Receiver	Requester	Client-connection	Server-connection
Transport type	√	O		O	√	
Transmission queue	√	√				
User ID	O	O		O	O	

Note: √ = Required attribute, O = Optional attribute

Channel functions

The channel functions available are shown in Table 47 on page 286. Here some more detail is given about the channel functions.

Create

You can create a new channel definition using the default values supplied by MQSeries, specifying the name of the channel, the type of channel you are creating, the communication method to be used, the transmission queue name and the connection name.

The channel name must be the same at both ends of the channel, and unique within the network. However, you should restrict the characters used to those that are valid for MQSeries object names.

Change

Use MQSC ALTER CHANNEL to change an existing channel definition, except for the channel name, or channel type.

Delete

Use MQSC DELETE CHANNEL to delete a named channel.

Display

Use MQSC DISPLAY CHANNEL to display the current definition for the channel.

Display Status

The MQSC DISPLAY CHSTATUS displays the status of a channel whether the channel is active or inactive. It applies to all message channels, but not to MQI channels. See "Display channel status" on page 290.

Information displayed includes:

- Channel name
- Communication connection name
- In-doubt status of channel (where appropriate)
- Last sequence number
- Transmission queue name (where appropriate)
- The in-doubt identifier (where appropriate)
- The last committed sequence number
- Logical unit of work identifier
- Process ID
- Thread ID (OS/2 and Windows NT only)

Ping

Use MQSC **PING CHANNEL** to exchange a fixed data message with the remote end. This gives some confidence to the system supervisor that the link is available and functioning.

Ping does not involve the use of transmission queues and target queues. It uses channel definitions, the related communication link, and the network setup.

It is available from sender and server channels, only. The corresponding channel is started at the far side of the link, and performs the start up parameter negotiation. Errors are notified normally.

The result of the message exchange is presented as Ping complete or an error message.

Ping with LU 6.2: When Ping is invoked, by default no USERID or password flows to the receiving end. If USERID and password are required, they can be created at the initiating end in the channel definition. If a password is entered into the channel definition, it is encrypted by MQSeries before being saved. It is then decrypted before flowing across the conversation.

Start

Use MQSC **START CHANNEL** for sender, server, and requester channels. It should not be necessary where a channel has been set up with queue manager triggering.

The **START CHANNEL** command is also used for receiver channels that have a disabled status. Starting a receiver channel that is in disabled status resets the channel and allows it to be started from the remote channel.

When started, the sending MCA reads the channel definition file and opens the transmission queue. A channel start-up sequence is executed, which remotely starts the corresponding MCA of the receiver or server channel. When they have been started, the sender and server processes await messages arriving on the transmission queue and transmit them as they arrive.

When you use triggering or run channels as threads, you will need to start the trigger process to monitor the initiation queue. Use the runmqchi command for this.

For OS/2 and Windows NT, MQSeries provides a listener process for TCP/IP, LU 6.2, and NetBIOS. This process listens for incoming channel startup requests and services them. The advantage of using this process is that each new channel is started as a thread rather than as a new process.

However, TCP/IP and LU 6.2 do provide other capabilities:

- For TCP/IP on OS/2, UNIX systems, and Digital OpenVMS inetd (or an equivalent TCP/IP service on OpenVMS) can be configured to start a channel. This will be started as a separate process.
- For LU 6.2 in OS/2, using Communications Manager/2 it is possible to configure the Attach Manager to start a channel. This will be started as a separate process.
- For LU 6.2 in UNIX systems, configure your SNA product to start the LU 6.2 responder process.

Channel functions

- For LU6.2 in Windows NT, using SNA Server you can use TpStart (a utility provided with SNA Server) to start a channel. This will be started as a separate process.
- For LU6.2 in Digital OpenVMS systems, use RUNMQLSR to start the LU6.2 responder process.

Use of the Start option always causes the channel to re-synchronize, where necessary.

For the start to succeed:

- Channel definitions, local and remote must exist.
- Transmission queue must exist, and have no other channels using it.
- MCAs, local and remote must exist.
- Communication link must be available.
- Queue managers must be running, local and remote.
- Message channel must not be already running.

A message is returned to the screen confirming that the request to start a channel has been accepted. For confirmation that the start command has succeeded, check the error log, or use DISPLAY CHSTATUS. The error logs are:

OS/2 and Windows NT

`\mqm\qmgrs\qmname\errors\AMQERR01.LOG` (for each queue manager called qmname)

`\mqm\qmgrs\@SYSTEM\errors\AMQERR01.LOG` (for general errors)

Note: On Windows NT, you still also get a message in the Windows NT application event log.

UNIX systems

`/var/mqm/qmgrs/qmname/errors/AMQERR01.LOG` (for each queue manager called qmname)

`/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG` (for general errors)

Digital OpenVMS

`MQS_ROOT:[MQM.QMGRS.QMNAME.ERRORS]AMQERR01.LOG` (for each queue manager called qmname)

`MQS_ROOT:[MQM.QMGRS.$SYSTEM.ERRORS]AMQERR01.LOG` (for general errors)

Stop

Use the MQSC STOP CHANNEL to request the channel to stop activity. Any channel type is disabled by this command. The channel will not start a new batch of messages until the operator starts the channel again. (For information about restarting stopped channels, see the *MQSeries Distributed Queuing Guide*.)

You can select the type of stop you require:

Stop quiesce example

```
STOP CHANNEL(QM1.TO.QM2) MODE(QUIESCE)
```

This command requests the channel to close down in an orderly way, the current batch of messages is completed, and the syncpoint procedure is carried out with the other end of the channel.

Note: If the channel is idle this command will not terminate a receiving channel.

Stop force example

```
STOP CHANNEL(QM1.TO.QM2) MODE(FORCE)
```

Normally, this option should not be used. It will terminate the channel process or thread. The channel will not complete processing the current batch of messages, and could, therefore, leave the channel in doubt. In general, it is recommended that operators use the quiesce stop option.

Reset

Use MQSC **RESET CHANNEL** to change the message sequence number. This command is available for any message channel, but not for MQI channels (client-connection or server-connection). The first message starts the new sequence the next time the channel is started.

If the command is issued on a sender or server channel, it will inform the other side of the change when the channel is restarted.

Resolve

Use MQSC **RESOLVE CHANNEL** to request a channel to commit or backout in-doubt messages. This is used when the other end of the link has terminated, and there is no prospect of it returning. Any outstanding unit of work needs to be resolved with either backout or commit. Backout restores messages to the transmission queue, while Commit discards them.

The **RESOLVE CHANNEL** command may be needed when messages are held in-doubt by a sender or server. The option accepts one of two parameters: **BACKOUT** or **COMMIT**.

The channel program does not try to establish a session with a partner. Instead, it determines the logical unit of work identifier (LUWID) which represents the in-doubt messages. It then issues, as requested, either:

- **BACKOUT** to restore the messages to the transmission queue; or
- **COMMIT** to delete the messages from the transmission queue.

For the resolution to succeed:

- The channel must be inactive
- The channel must be in doubt
- The channel type must be sender or server
- A local channel definition must exist
- Local queue manager must be running

Channel functions

Appendix K. Setting up communication in Digital OpenVMS systems

Distributed queue management (DQM) is a remote queuing facility for MQSeries. It provides channel control programs for the queue manager that form the interface to communication links, controllable by the system operator. The channel definitions held by distributed queue management use these connections.

When a distributed queue management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This chapter explains how to do this.

Deciding on a connection

There are four forms of communication for MQSeries on Digital OpenVMS systems:

- TCP/IP
- LU 6.2
- DECnet Phase IV
- DECnet Phase V

Each channel definition must specify one only as the transmission protocol (Transport Type) attribute. One or more protocols may be used by a queue manager.

For MQSeries clients, it may be useful to have alternative channels using different transmission protocols. There is an example of this in the *MQSeries Clients* book.

Defining a TCP/IP connection

The channel definition at the sending end specifies the address of the target. The TCP/IP service is configured for the connection at the receiving end.

Sending end

Specify the host name, or the TCP/IP address of the target machine, in the Connection Name field of the channel definition. Port number 1414 is assigned by the Internet Assigned Numbers Authority to MQSeries.

To use a port number other than the default, change the connection name field thus:

```
Connection Name REMHOST(1822)
```

where REMHOST is the hostname of the remote machine and 1822 is the port number required. (This must be the port that the listener at the receiving end is listening on.)

Alternatively you can change the default sending port number by specifying it in the queue manager configuration file (qm.ini):

```
TCP:
  Port=1822
```

Defining a TCP/IP connection

Using the TCP/IP SO_KEEPALIVE option

If you want to use the SO_KEEPALIVE option you must add the following entry to your queue manager configuration file (QM.INI):

```
TCP:
  KeepAlive=yes
```

Receiving channels using Digital TCP/IP services (UCX) for OpenVMS

To use Digital TCP/IP Services (UCX) for OpenVMS, you must configure a UCX service as follows:

1. Create a file consisting of one line and containing the DCL command to start the TCP/IP receiver program, amqcrsta.exe:

```
$ mcr amqcrsta [-m Queue_Man_Name]
```

Place this file in the SYS\$MANAGER directory. In this example the name of the file is MQRECV.COM.

Notes:

- a. If you have multiple queue managers you must make a new file and UCX service for each queue manager.
 - b. Ensure that the protection on the file and its parent directory allow it to be executable, that is, the protection is /PROT=W:RE.
2. Create a UCX service to start the receiving channel program automatically:

```
$ UCX
UCX> set service MQSeries/port=1414/protocol=TCP/user_name=MQM -
UCX> /process=MQSERIES/file=SYS$MANAGER:MQRECV.COM/limit=6
UCX> enable service MQSeries
UCX> exit
```

If a receiving channel does not start when the sending end starts, it is probably due to the permissions on the file being incorrect.

3. To enable the service upon every system IPL (reboot), issue the command

```
$ UCX SET CONFIGURATION ENABLE SERVICE MQSERIES
```

Receiving channels using Cisco MultiNet for OpenVMS

To use Cisco MultiNet for OpenVMS, you must configure a Multinet service as follows:

1. Create a file consisting of one line and containing the DCL command to start the TCP/IP receiver program, amqcrsta.exe:

```
$ mcr amqcrsta.exe [-m Queue_Man_Name]
```

Place this file in the SYS\$MANAGER directory.

Notes:

- a. If you have multiple queue managers you must make a new file and MultiNet service for each queue manager.
- b. Ensure that the protection on the file and its parent directory allow it to be executable, that is, the protection is /PROT=W:RE.

2. Create a MultiNet service to start the receiving channel program automatically:

```
$ multinet configure/server
MultiNet Server Configuration Utility 3.5 (101)
[Reading in configuration from MULTINET:SERVICES.MASTER_SERVER]
SERVER-CONFIG> add MQSeries
[Adding new configuration entry for service "MQSERIES"]
Protocol: [TCP]
TCP Port number: 1414
Program to run: sys$manager:mqrecv.com
[Added service MQSERIES to configuration]
[Selected service is now MQSERIES]
SERVER-CONFIG> set flags UCX_SERVER
MQSERIES flags set to <UCX_SERVER>]
SERVER-CONFIG> set username MQM
[Username for service MQSERIES set to MQM]
SERVER-CONFIG> exit
[Writing configuration to MULTINET_COMMON_ROOT:SERVICES.MASTER_SERVER]
$
```

The service is enabled automatically after the next system IPL (reboot). To enable the service immediately, issue the command

```
'MULTINET CONFIGURE /SERVER RESTART'.
```

Receiving channels using Attachmate** PathWay for OpenVMS

To use Attachmate PathWay for OpenVMS to start channels, you *must* configure a PathWay service as follows:

1. Create a file consisting of one line and containing the DCL command to start the TCP/IP receiver program, amqcrsta.exe:

```
$ mcr amqcrsta [-m Queue_Manager_Name]
```

Place this file in the SYS\$MANAGER directory. In this example the name mqrecv.com is used.

2. Create an Attachmate service to start the receiving channel program automatically.

You do this by adding the following lines to the file TWG\$COMMON:[NETDIST.ETC]SERVERS.DAT.

```
# MQSeries
service-name    MQSeries
program         SYS$MANAGER:MQRECV.COM
socket-type     SOCK_STREAM
socket-options  SO_ACCEPTCONN | SO_KEEPALIVE
socket-address  AF_INET , 1414
working-set     512
priority        4
INIT            TCP_Init
LISTEN         TCP_Listen
CONNECTED      TCP_Connected
SERVICE       Run_Program
username       MQM
device-type     UCX
```

Defining an LU 6.2 connection

MQSeries for Digital OpenVMS uses the DECnet SNA APPC/LU 6.2 Programming Interface. This interface requires access through DECnet to a suitably configured SNA Gateway, for example, the SNA Gateway-ST, or SNA Gateway-CT.

SNA configuration

To enable MQSeries to work with DECnet APPC/LU6.2 you **must** complete your Gateway SNA configuration first. The DEC SNA configuration **must** be in agreement with the Host SNA configuration.

Notes:

1. When configuring your host system, be aware that the VMS SNA Gateway supports PU 2 and **not** Node type 2.1. This means that the LUs configured on the mainframe for the DEC **must** be dependent LUs and that the mode specified must specify a single session.
2. Ensure that the SNA libraries are installed as shared images upon each system IPL by running the command @SYS\$STARTUP:SNALU62\$STARTUP.COM in the system startup procedure.

To configure your SNA Gateway, set up the SNAGATEWAY_<node>_SNA.COM file, where <node> is replaced with the node name of your DECnet SNA gateway.

Do this by responding to the configuration prompts in the Gateway installation procedure, or by directly editing the file.

The SNA Gateway installation procedure creates the file in the directory SYS\$COMMON:[SNA\$CSV].

The configuration information in this file is downloaded to the Gateway when you run the NCP LOAD NODE command.

Notes:

1. SNANCP commands can be used to make online changes to the current Gateway configuration.
2. SNAP can be used to monitor SNA resources.

A sample SNA configuration follows:

```
#!+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
$! Start of file: SYS$COMMON:[SNA$CSV]SNAGATEWAY_SNAGWY_SNA.COM
$! DECnet SNA Gateway-ST SNA configuration file
$! Created: 23-FEB-1996 19:10:43.68 by SNACST$CONFIGURE V1.2
$! Host node: CREAMP User$ CHO
#!+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
$ v = f$verify(1)
$ RUN SYS$SYSTEM:SNANCP
SET LINE SYN-0 -
    DUPLEX FULL -
    PROTOCOL SDLC POINT -
    SIGNALLING NORMAL -
    CLOCK EXTERNAL -
    MODEM TYPE NORMAL -
    RECEIVE BUFFERS 34 -
    LOGGING INFORMATIONAL -
```

```

BUFFER SIZE 265
SET CIRCUIT SDLC-0 -
  LINE SYN-0 -
  DUPLEX FULL -
  RESPONSE MODE NORMAL -
  STATION ADDRESS C1 -
  LOGGING INFORMATIONAL -
  STATION ID 0714002A          // XID
SET PU SNA-0 CIRCUIT SDLC-0 -
  LU LIST 1-32 -
  SEGMENT SIZE 265 -          // must equal MAXDATA on Host PU definition
  LOGGING WARNING
SET CIRCUIT SDLC-0 STATE ON
SET LINE SYN-0 STATE ON
SET SERVER SNA-ACCESS -
  LOGGING WARNING -
  NOTE "Gateway Access Server" -
  STATE ON
SET ACCESS NAME VTAMSDR PU SNA-0 LU 2 APPL IYZIZCB1 LOGON MQMODE
SET ACCESS NAME VTAMRQST PU SNA-0 LU 3 APPL IYZIZCB1 LOGON MQMODE
SET ACCESS NAME VTAMSVR PU SNA-0 LU 4 APPL IYZIZCB1 LOGON MQMODE
SET ACCESS NAME VTAMRCVR PU SNA-0 LU 5 APPL IYZIZCB1 LOGON MQMODE
$ EXIT $STATUS + (0 * 'f$verify(v)')
$!+-----+
$! End of file: SYS$COMMON:[SNA$CSV]SNAGATEWAY_SNAGWY_SNA.COM
$!+-----+

```

Defining access names

You should set up a separate Access name for each MQSeries channel. This ensures that the VMS system and the remote system agree on the LU used for the channel. If you use a single access name, with a range of LUs specified, the Gateway selects the LUs in a circular order. Therefore the LU selected by the Gateway will not correspond with the LU used by the Host channel, because the Host associates a specific LU with a channel.

The access name is used only to communicate between the DECnet SNA APPC program and the Gateway. It has no network meaning.

Notes:

1. The LUs are single session. You **must** define a separate LU for requester and sender channel pairs.
2. You are advised to use names that associate the access name to the corresponding channel, but you can choose any name.
3. The APPL is the ACBNAME in the VBUILD statement defining the APPL in VTAM.
4. The LU must correspond to the LOCADDR in the LU definition statement in VTAM.
5. The LOGON must specify the logmode entry on the host that specifies parameters acceptable to the SNA Gateway.

The *DECnet SNA Gateway Guide to IBM Parameters* details the parameters expected by the Gateway.

Specifying SNA configuration parameters to MQSeries

MQSeries obtains knowledge of the SNA resources by passing the Gateway Node name and the Access name to the channel program.

Passing parameters to sender and requester channel pairs

For sender and requester channel pairs specify the Gateway Node and Access Name in the CONNAME string in the channel definition.

The CONNAME also includes the TPNAME that is used by the SNA Allocate verb to invoke the remote program.

The format of the CONNAME is: `CONNAME('GatewayNode.AccessName(TpName)')`.

For example: `CONNAME('SNAGWY.VMSREQUESTER(HOSTSVR)')`,

where SNAGWY is the Gateway node, VMSREQUESTER is the access name, and HOSTSVR is the TPNAME.

The TPNAME must be agreed on by the OpenVMS network manager and the Host. MQSeries does not require a specific name.

Running senders and requesters

Senders, requesters, and fully qualified servers can be explicitly run by performing a START CHANNEL command in runmqsc.

Senders and requesters on Digital OpenVMS initiate a session by issuing an INIT-SELF to request a BIND from the host. In issuing the Allocate verb, the MQSeries channel program takes the LU name and the Mode Name from the Access Name.

MQSeries then allocates a conversation using the specified TPNAME.

Passing parameters to servers and receivers

For servers and receivers, specify the Gateway Node, Access Name, and TPNAME as command line parameters to the `runmqslsr` command.

Running servers and receivers

Servers and receivers are started by running the `runmqslsr` command.

```
$ RUNMQLSR -m QMname -n TPname -g GatewayNode(AccessName)
```

Note: Each server and receiver channel requires its own listener process.

You can include these commands in the MQSeries startup file, `SYS$STARTUP:MQS_STARTUP`.

Receivers and servers issue the `ACTIVATE_SESSION` request to the Gateway in passive mode. In passive mode the channel program waits for a BIND from the remote system, which puts the LU into the active-listening state, waiting for a bind from the host.

You can check the LU status using `SNANCP` to make sure that you are in active-listening state on the correct LU. If a BIND from the host arrives specifying the LU that is in active-listening state, the session will be established. After establishing the session, the host attempts to allocate a conversation.

The TPNAME used by the host **must** be the same name as that specified in the command line to establish the conversation.

Ending the SNA Listener process

To find the batch job number for the SNA listener process, type: `$ show queue / all`

To end the SNA Listener process type:

```
$ delete /entry=<jobnumber>
```

where <jobnumber> is the job number of the listener batch job.

Sample MQSeries configuration

```
*
*      channel configuration for saturn.queue.manager for LU6.2
*
def q1('HOST_SENDER_TQ') usage(xmitq)

def q1('HOST_SERVER_TQ') usage(xmitq)

def chl('HOST.TO.VMS') chltype(rcvr) trptype(lu62) +
  seqwrap(999999999)

def chl('VMS.TO.HOST') chltype(svr) trptype(lu62) +
  xmitq('HOST_SERVER_TQ') seqwrap(999999999)

def chl('VMS.TO.VTAM') chltype(sdr) trptype(lu62) +
  conname('SNAGWY.VMSSENDER(HOSTRCV)') +
  xmitq('HOST_SENDER_TQ') seqwrap(999999999)

def chl('VTAM.TO.VMS') chltype(rqstr) trptype(lu62) +
  conname('SNAGWY.VMSREQUESTER(HOSTSVR)') seqwrap(999999999)
```

In this example four channels, one for each channel type – sender, requester, server, and receiver – have been set up.

On the remote system you need to configure four corresponding channels. Channels that talk to each other must have the same name.

- The OpenVMS sender, VMS.TO.VTAM, talks to a receiver called VMS.TO.VTAM on the host.
- The OpenVMS requester, VTAM.TO.VMS talks to a server VTAM.TO.VMS on the host.
- The OpenVMS server, VMS.TO.HOST talks to a requester VMS.TO.HOST on the host.
- The OpenVMS receiver, HOST.TO.VMS talks to a sender HOST.TO.VMS on the Host.

The commands to start each channel are:

```
$ runmqchl -m "saturn.queue.manager" -c "VTAM.TO.VMS"
$ runmqchl -m "saturn.queue.manager" -c "VMS.TO.VTAM"
$ runmqlsr -m "saturn.queue.manager" -n "HOSTSDR" -g SNAGWY(VMSRECEIVER)
$ runmqlsr -m "saturn.queue.manager" -n "HOSTRQSTR" -g SNAGWY(VMSSENDER)
```

Problem solving

Error PUNOTAVA - PU has not been activated

This error indicates a lack of connectivity between the two machines. Make sure your line and circuit are set to state ON. Use SNATRACE at the circuit level to verify that the Digital OpenVMS machine is polling. If no response is received for the poll, check that the PU on the host is enabled. If the line will not go to the ON STATE check your physical line. If the trace shows the host responding to the poll, but the PU still does not become active, check your setting of the STATION ID.

Failure to allocate conversation

This error is returned by a sender or requester to indicate that allocate failed. Run trace to verify that the session can be established. Verify that the Digital OpenVMS machine sends the INIT-SELF (010681). If there is no response to the INIT-SELF make sure that the host MQSeries channel is started. If the BIND from the host is rejected by the Digital OpenVMS machine analyze the DEC bind response. Use the *Guide to IBM Parameters* to see what is set incorrectly in the mode. If a session is established and the conversation allocate request is rejected verify that the TPNAMEs are configured the same on both systems.

For receivers and servers verify that a BIND is sent by the host. If not, enable the Host MQSeries channel. If the BIND is rejected check the reason for rejection. Make sure that the Digital OpenVMS listener LU is the LU with which the host is trying to establish a session.

MQSeries connection failure

After establishing a conversation the two MQSeries channels engage in a protocol to establish an MQSeries channel connection. If this fails, the reason for failure should be indicated in the error logs on the two systems. Check both logs and correct the indicated problem. For example the connection fails if one system has a SEQWRAP value of 999999999 and the other 999999. In the SNATRACE you will see that the allocate succeeded and that MQ is trying to establish a channel connection. At this point the MQSeries logs are the best aid in resolving problems.

Appendix L. Defining DECnet Phase IV and Phase V connections

Note

The information in this appendix will be inserted into the *MQSeries Distributed Queuing Guide* the next time that this book is refreshed.

Defining a DECnet Phase IV connection

The channel definition at the sending end specifies the address of the target. The DECnet network object is configured for the connection at the receiving end.

Sending end

Specify the DECnet node name and the DECNET object name in the Connection Name field of the channel definition. You need a different DECnet object for each separate queue manager that is defined. For example, to specify DECnet object MQSERIES on node FOONT enter the following when defining the channel:

```
CONNAME('FOONT(MQSERIES)')
```

Receiving on DECnet Phase IV

To use DECnet Phase IV to start channels, you must configure a DECnet object as follows:

1. Create a file consisting of one line and containing the DCL command to start the DECnet receiver program, amqcrsta.exe:

```
$ mcr amqcrsta [-m Queue_Man_Name] -t DECnet
```

Place this file in the SYSS\$MANAGER directory. In this example the file is named MQRECVDECNET.COM.

Notes:

- a. If you have multiple queue managers you **must** make a new file and DECnet object for each queue manager.
 - b. If a receiving channel does not start when the sending end starts, it is probably due to the permissions on this file being incorrect.
2. Create a DECnet object to start the receiving channel program automatically. You must supply the correct password for MQSeries.

```
$ MCR NCP
NCP> define object MQSERIES
Object number      (0-255): 0
File name          (filename):sys$manager:mqrecvdecnet.com
Privileges (List of VMS privileges):
Outgoing connect privileges (List of VMS privileges):
User ID            (1-39 characters): mqm
Password           (1-39 characters): mqseries
Account            (1-39 characters):
Proxy access (INCOMING, OUTGOING, BOTH, NONE, REQUIRED):
NCP> set known objects all
NCP> exit
```

Note: You could use proxy user identifiers rather than actual user identifiers. This will prevent any unauthorized access to the database. Information on how to set up proxy identifiers is given in the *Digital DECnet for OpenVMS Networking Manual*.

3. Ensure that all known objects are set when DECnet is started.

Defining a DECnet Phase V connection

Set up the MQSeries configuration for channel objects:

1. Start the NCL configuration interface by issuing the following command:

```
$ MC NCL
NCL>
```

2. Create a session control application entity by issuing the following commands:

```
NCL> create session control application MQSERIES
NCL> set sess con app MQSERIES address {name=MQSERIES}
NCL> set sess con app MQSERIES image name -
_ SYSS$MANAGER:MQRECVDECNET.COM
NCL> set sess con app MQSERIES user name "MQM"
NCL> set sess con app MQSERIES node synonym true
NCL> show sess con app MQSERIES all [characteristics]
```

Note: User-defined values are in **uppercase**.

3. Create the command file as for DECnet Phase IV.
4. The log file for the object is net\$server.log in the sys\$logdir directory for the application-specified user name.
5. To enable the session control application upon every system IPL (reboot), add the preceding NCL commands to the file SYSS\$MANAGER:NET\$APPLICATION_LOCAL.NCL.

Appendix M. Ancillary information

This appendix lists any ancillary information that you need to setup MQSeries for Digital OpenVMS.

The information contained in this appendix will be inserted into the identified book, the next time that the book is refreshed.

MQSeries Command Reference

The TRPTYPE for DECnet is DECNET. This change applies to the ALTER CHANNEL, DEFINE CHANNEL, and DISPLAY CHANNEL commands.

MQSeries Distributed Queuing Guide

The TRPTYPE for DECnet is DECNET when setting up channels.

The channel program amqcrsta supports DECnet in addition to TCP/IP. See Appendix L, "Defining DECnet Phase IV and Phase V connections" on page 305 for further information.

Ancillary information

Appendix N. Messages

This appendix describes the format of the messages issued by MQSeries and how they are documented.

Message format

The format of the MQSeries messages is as follows:

- The message identifier, where the identifier has two components:
 1. The characters "AMQ," which identify the message as originating from MQSeries
 2. A four-digit decimal code.
- Text of the message

Structure of messages

This section describes the structure of MQSeries messages.

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*.

In this book, the message variables are shown as an '&' symbol, followed by a number.

Where there is more than one variable in a message, a different number is added to each '&' symbol.

Note: You should always look at the extended help for a message before carrying out any other action, because, in certain cases, the variables are displayed in the extended help only.

Message information

Where applicable, this information is also provided:

Explanation: Why the message was issued.

User action: Instructions to the user.

Note: The message file may contain the explanation of the message, in addition to the message itself.

MQSeries messages

MQSeries messages are numbered 5000 through 9999, and they are listed in this book in numeric order. However, not all numbers have been used, and therefore, the list is not continuous.

Message groups

MQSeries messages are grouped according to the part of MQSeries from which they originate:

5000 through 5999 Installable services – see page 311.

6000 through 6999 Common services – see page 315.

7000 through 7999 The MQSeries product – see page 318.

8000 through 8999 Administering MQSeries – see page 326.

9000 through 9999 Remote – see page 337.

Installable services messages

AMQ5006 Unexpected error: rc = &1

Explanation: An unexpected error occurred in an internal function of the product.

User action: Save the generated output files and contact your IBM support center.

AMQ5501 There was not enough storage to satisfy the request

Explanation: An internal function of the product attempted to obtain storage, but there was none available.

User action: Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your IBM support center.

AMQ5511 Installable service component '&3' returned '&4'.

Explanation: The internal function, that adds a component to a service, called the component initialization process. This process returned an error.

User action: Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5512 Installable service component '&3' returned '&4' for queue manager name = '&5'.

Explanation: An installable service component returned an unexpected return code.

User action: Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5513 '&3' returned &1.

Explanation: An unexpected error occurred.

User action: Save the generated output files and contact your IBM support center.

AMQ5600 Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

AMQ5603 Usage: dltmqm [-z] QMgrName

AMQ5604 Usage: dspmqaut [-m QMgrName] [-n ObjName] [-t ObjType] [-p Principal] [-g Group] [-s ServiceName]

AMQ5605 Usage: endmqm [-z] [-c | -i | -p] QMgrName

AMQ5606 Usage: setmqaut -m QMgrName [-n ObjName] [-t ObjType] [-p Principal] [-g Group] [-s ServiceName] Authorizations

AMQ5607 Usage: strmqm [-z] [QMgrName]

AMQ5608 Usage: dspmqtrn QMgrName

AMQ5609 Usage: rsvmqtrn -m QMgrName (-c | -b)

Transaction,Number

AMQ5700 Queue manager name '&3', work queue name '&4'.

Explanation: These are the values of the parameters with which the add-in task was started.

User action: None.

Programmer response: None.

AMQ5701 Checking mail-in database &3

Explanation: The add-in task is performing a periodic check for mail memos that have arrived in the mail-in database called &3.

User action: None.

Programmer response: None.

AMQ5702 Checking for replies.

Explanation: The add-in task is checking the reply queues for responses from MQSeries applications.

User action:

Programmer response: None.

AMQ5703 MQSeries add-in task ended.

Explanation: Termination of the MQSeries add-in task has completed.

User action: None.

Programmer response: None.

AMQ5704 Terminating.

Explanation: The add-in task is terminating, either due to a user request or an error.

User action: None.

Programmer response: None.

AMQ5705 Initializing.

Explanation: The add-in task is initializing. It processes the link database and connects to the queue manager in preparation to receive requests.

User action: None.

Programmer response: None.

AMQ5706 Mail-in database '&3', link database '&4', wait time &1 seconds.

Explanation: These are the values of the parameters with which the add-in task was started.

User action: None.

Programmer response: None.

AMQ5707 Add-in task initialization complete.

Explanation: The add-in task has finished reading the link database and is now ready to process requests.

User action: None.

Programmer response: None.

AMQ5708 Only two-byte integer values are supported for S390 format.

Explanation: The add-in task supports conversion of two-byte integers from S390 systems.

User action: Ensure that the entry in the link database uses fields of only two bytes in length if they are in the S390 format.

Programmer response: None.

AMQ5710 Text of user document causing previous message: '&3'.

Explanation: The add-in task generated the previous message in response to an error. This message contains the text of the user note associated with the error.

User action: None.

Programmer response: None.

AMQ5711 An error occurred in reading the link database.

Explanation: The add-in task detected an error while reading the link database.

User action: Use the information in previous error messages to diagnose the error. Then, correct the contents of the link database and restart the add-in task.

Programmer response: None.

AMQ5712 An error occurred while setting field '&5' in user document, return code &3

Explanation: The add-in task was trying to update a document in response to a reply from an MQSeries application. An error was encountered during the update of the field '&5'. The link database entry '&4' was being used to perform the update.

User action: Make sure that the entry in the link database matches the description of the form being used for the update.

Programmer response: None.

AMQ5714 Field '&4' not found in link database entry.

Explanation: The add-in task could not find a field called '&4' during processing of the link database. This field is a required field.

User action: Examine the definition of the link database being used to ensure that all of the required fields are supplied. Refer to the IBM-supplied sample link database for an example of a valid link database.

Programmer response: None.

AMQ5715 Data type '&4' not supported.

Explanation: The add-in task does not support the data type '&4'.

User action: Consult the MQSeries documentation for a description of the list of supported data types. Update the entry in the link database using the unsupported data type. Then, stop and restart the add-in task.

Programmer response: None.

AMQ5716 An error occurred connecting to MQSeries queue manager '&4', reason code &3

Explanation: The add-in task could not connect to MQSeries queue manager '&4'. The reason code from MQCONN was &3.

User action: Look up the reason code in the MQSeries documentation to establish the cause of the error. Ensure that the queue manager exists and is running. If the add-in task is running as an MQSeries client, ensure that it can communicate with the server queue manager.

Programmer response: None.

AMQ5717 An error occurred disconnecting from MQSeries queue manager '&4', return code '&3'.

Explanation: The add-in task encountered an error disconnecting from the MQSeries queue manager '&4'. The reason code from MQDISC was &3.

User action: Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response: None.

AMQ5718 An error occurred during processing of a request in the mail-in database.

Explanation: The add-in task encountered an error during processing of a request in the mail-in database. The processing involves transforming the contents of the mail memo into a message which is placed on an MQSeries queue. If the message has a reply, an additional message is formatted and placed on the internal work queue.

User action: Use the information in previous error messages to diagnose the error.

Programmer response: None.

AMQ5720 Errors detected in response message from MQSeries application.

Explanation: The response from an MQSeries application to a message sent by the add-in task satisfied the error conditions specified in the corresponding link database entry. The error data is '&4'.

User action: Examine the error conditions in the link database entry to establish why the error conditions were satisfied. If an invalid request message was sent to the MQSeries application, correct the request messages being sent. If the problem was due to an error encountered by the MQSeries application, correct the cause of the error and retry the request.

Programmer response: None.

AMQ5721 An error occurred opening internal work file '&4'.

Explanation: The add-in task could not open the internal work file used to hold the contents of a mail memo during processing. Possible causes include more than one program trying to use the same file.

User action: Ensure that there is only one copy of the MQSeries add-in task running.

Programmer response: None.

AMQ5723 Memory allocation failed.

Explanation: The add-in task was unable to allocate storage.

User action: Try to free up some system memory and retry the operation.

Programmer response: None.

AMQ5725 Empty mail memo received from mail-in database.

Explanation: The add-in task found a mail memo with an empty body in the mail-in database. Mail memos in the mail-in database must contain the information required to generate a message to place on an MQSeries queue.

User action: Ensure that all entries placed in the mail-in database have the expected contents. None.

Programmer response: None.

AMQ5727 Link database entry '&4' cannot be found.

Explanation: The add-in task received a request without a corresponding entry in the link database. The name of the required entry is '&4'.

User action: Either add an entry of the correct name to the link database or change the request being generated to use an existing entry in the link database. If you add an entry to the link database, you will have to stop and restart the add-in task before the change takes effect.

Programmer response: None.

AMQ5729 An error was encountered by the add-in task.
Check the mail for details.

Explanation: This message is inserted into the error_field_msg field of a user document if an error is encountered by the add-in task during the processing of the document's associated mail memo.

User action: None.

Programmer response: None.

AMQ5730 Error encountered by MQSeries add-in task

Explanation: This is the subject line of mail memos sent by the add-in task.

User action: None.

Programmer response: None.

AMQ5731 Idle.

Explanation: The add-in task is waiting for the configured time interval to elapse before checking the mail-in database for new requests and checking the reply queues for new replies.

User action: None.

Programmer response: None.

AMQ5732 LOAD MQLINK -t" -q WorkQName" -w WaitTime" -d MailInDB" -I LinkDB" QMgrName"

Explanation: This is a summary of the correct syntax for invoking the MQSeries add-in task in Lotus Notes. If you specify a queue manager name, it must be the last parameter. The order of the other parameters is not significant.

User action: None.

Programmer response: None.

AMQ5733 MQSeries add-in task loading.

Explanation: The add-in task has been started and is accessing the link database in preparation to receive requests.

User action: None.

Programmer response: None.

AMQ5734 An error occurred opening the database '&4'. The error code was &3.

Explanation: The add-in task could not open the named database. This could be because the database does not exist.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5735 An error occurred opening the mail file '&4'. The error code was &3.

Explanation: The add-in task could not open the named mail file.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5736 An error occurred searching the database '&4'. The error code was &3.

Explanation: The add-in task could not search the named database.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5737 An error occurred deleting an entry from the database '&4'. The error code was &3.

Explanation: The add-in task could not delete an entry from the named database.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5738 An error occurred extracting the contents of a mail memo in the mail-in database '&5' to the file called '&4'. The error code was &3.

Explanation: The add-in task could not extract the body of a mail memo into the named file. Possible causes include being unable to create the file or another program already using the file.

User action: Ensure that there is only one copy of the MQSeries add-in task running. If the problem was due to the configuration in which you are operating Lotus Notes, refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5739 An error occurred opening a mail memo in the mail-in database '&4'. The error code was &3.

Explanation: The add-in task could not open a mail memo in the named mail-in database.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5740 An error occurred opening an entry in the link database '&4'. The error code was &3.

Explanation: The add-in task could not open an entry in the link database.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5741 An error occurred creating a mail memo. The error code was &3.

Explanation: The add-in task could not create a mail memo. This is probably due to a shortage of resources.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5742 Could not send a mail memo to user '&4'. The error code was &3.

Explanation: The add-in task could not send a mail memo to the named user to report an error condition.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5743 Could not find entry with ID '&5' in database '&4'. The error code was &3.

Explanation: The add-in task could not find an entry in the database '&4' which it was to update in response to a reply from an MQSeries application. This may indicate that the entry has been manually deleted or that another application has already updated the entry.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5744 Could not update an entry in database '&4'. The error code was &3.

Explanation: The add-in task could not update an entry in the database '&4' in response to a reply from an MQSeries application.

User action: Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response: None.

AMQ5745 An error occurred opening MQSeries queue '&4', reason code &3.

Explanation: The add-in task could not open MQSeries queue '&4'. MQOPEN was called with open options &5. The reason code from MQOPEN was &3.

User action: Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response: None.

AMQ5746 An error occurred putting a message on MQSeries queue '&4', reason code &3.

Explanation: The add-in task could not put a message on MQSeries queue '&4'. The reason code from MQPUT was &3.

User action: Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response: None.

AMQ5747 An error occurred getting a message from MQSeries queue '&4', reason code &

Explanation: The add-in task could not get a message from MQSeries queue '&4'. The reason code from MQGET was &3.

User action: Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response: None.

Common services messages

AMQ6004 An error occurred during MQSeries initialization or ending.

Explanation: An error was detected during initialization or ending of MQSeries. The MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6025 Program not found.

Explanation: MQSeries is unable to start program &3 because it was not found.

User action: Check the program name is correctly specified and rerun the program.

AMQ6026 A resource shortage prevented the creation of an MQSeries process.

Explanation: An attempt to create an MQSeries process was rejected by the operating system due to a process limit (either the number of processes for each user or the total number of processes running system wide), or because the system does not have the resources necessary to create another process.

User action: Investigate if a process limit is preventing the creation of the process and if so why the system is constrained in this way. Consider raising this limit or reducing the workload on the system.

AMQ6035 MQSeries failed, no storage available.

Explanation: An internal function of the product attempted to obtain storage, but there was none available.

User action: Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your IBM support center.

AMQ6037 MQSeries was unable to obtain enough storage.

Explanation: The product is unable to obtain enough storage. The product's error recording routine may have been called.

User action: Stop the product and restart it. If this does not resolve the problem see if a problem has been recorded. If a problem has been recorded, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6047 Conversion not supported.

Explanation: MQSeries is unable to convert string data tagged in CCSID &1 to data in CCSID &2.

User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6048 DBCS error

Explanation: MQSeries is unable to convert string data due to a DBCS error. Conversion is from CCSID &1 to CCSID &2.

User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6049 DBCS only string not valid.

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2. Message descriptor data must be in single byte form. CCSID &2 is a DBCS only CCSID.

User action: Check the CCSID of your job or system and change it to one supporting SBCS or mixed character sets. Refer to the appropriate National Language Support publications for character sets and CCSIDs supported.

AMQ6050 CCSID error.

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6051 Conversion length error.

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, due to an input length error.

AMQ6052 Conversion length error.

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

AMQ6053 CCSID error

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

User action: One of the CCSIDs is not supported by the system. Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6064 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6090 MQSeries was unable to display an error message.

Explanation: MQSeries has attempted to display the message associated with return code &6. The return code indicates that there is no message text associated with the message. Associated with the request are inserts &1 : &2 : &3 : &4 : &5.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6091 An internal MQSeries error has occurred.

Explanation: Private memory has detected an error, and is abending due to &3. The error data is &1.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6100 An internal MQSeries error has occurred.

Explanation: MQSeries has detected an error, and is abending due to &3. The error data is &1.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6107 CCSID not supported.

Explanation: MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, because one of the CCSIDs is not recognized.

User action: Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6115 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6118 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6119 An internal MQSeries error has occurred.

Explanation: MQSeries detected an unexpected error when calling the operating system. The MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6120 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6121 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: MQSeries has detected a parameter count of &1 that is not valid. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6122 An internal MQSeries error has occurred.

Explanation: An error has been detected, and the MQSeries error recording routine has been called.

User action: MQSeries has detected parameter &1 that is not valid, having value &2&3. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6125 An internal MQSeries error has occurred.

Explanation: An internal error has occurred with identifier &1. This message is issued in association with other messages.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6148 An internal MQSeries error has occurred.

Explanation: MQSeries has detected an error, and is abending due to &3. The error data is &1.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6172 No codeset found for current locale.

Explanation: No codeset could be determined for the current locale. Check that the locale in use is supported.

User action: None.

AMQ6173 No CCSID found for codeset &3.

Explanation: Codeset &3. has no supported CCSID. Check that the locale in use is supported. CCSIDs can be added by updating the file /var/mqm/conv/table/ccsid.tbl.

User action: None.

AMQ6708 A disk full condition was encountered when formatting a new log file in location &3.

Explanation: The queue manager attempted to format a new log file in directory &3. The drive or file system containing this directory did not have sufficient free space to contain the new log file.

User action: Increase the amount of space available for log files and retry the request.

AMQ6710 Queue manager unable to access directory &3.

Explanation: The queue manager was unable to access directory &3 for the log. This could be because the directory does not exist, or because the queue manager does not have sufficient authority.

User action: Ensure that the directory exists and that the queue manager has authority to read and write to it. Ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ6767 Log file &3 could not be opened for use.

Explanation: Log file &3 could not be opened for use. Possible reasons include the file being missing, the queue manager being denied permission to open the file or the contents of the file being incorrect.

User action: If the log file was required to start the queue manager, ensure that the log file exists and that the queue manager is able to read from and write to it. If the log file was required to recreate an object from its media image and you do not have a copy of the required log file, delete the object instead of recreating it.

MQSeries product messages

AMQ7001 The location specified for creation of the queue manager is not valid.

Explanation: The directory under which queue managers are to be created is not valid. It may not exist, or there may be a problem with authorization.

User action: The location is specified in the machine-wide ini file. Correct the file and submit the request again.

AMQ7002 An error occurred manipulating a file.

Explanation: An internal error occurred while trying to create or delete a queue manager file. It is likely that the error was caused by there being insufficient space on a disk, or by problems with authorization to the underlying filesystem.

User action: Identify the file that caused the error, using problem determination techniques. Correct the error in the filesystem and submit the request again.

AMQ7005 The queue manager is running.

Explanation: You tried to perform an action that requires the queue manager stopped, however, it is currently running. You probably tried to delete or start a queue manager that is currently running.

User action: If the queue manager should be stopped, stop the queue manager and submit the failed command again.

AMQ7006 Missing attribute &5 on stanza starting on line &1 of ini file &3.

Explanation: The &4 stanza starting on line &1 of configuration file &3 is missing the required &5 attribute.

User action: Check the contents of the file and retry the operation.

AMQ7008 The queue manager already exists.

Explanation: You tried to create a queue manager that already exists.

User action: If you specified the wrong queue manager name, correct the name and submit the request again.

AMQ7010 The queue manager does not exist.

Explanation: You tried to perform an action against a queue manager that does not exist. You may have specified the wrong queue manager name.

User action: If you specified the wrong name, correct it and submit the command again. If the queue manager should exist, create it, and then submit the command again.

AMQ7012 The specified trigger interval is not valid.

Explanation: You specified a value for the trigger interval that is not valid. The value must be not less than zero and not greater than 999 999 999.

User action: Correct the value and resubmit the request.

AMQ7013 There is an error in the name of the specified dead letter queue.

Explanation: You specified a name for the dead letter queue that is not valid.

User action: Correct the name and resubmit the request.

AMQ7014 There is an error in the name of the specified default transmission queue.

Explanation: You specified a name for the default transmission queue that is not valid.

User action: Correct the name and submit the command again.

AMQ7015 There is an error in the maximum number of open object handles specified.

Explanation: You specified a value for the maximum number of open object handles to be allowed that is not valid. The value must be not less than zero and not greater than 999 999 999.

User action: Correct the value and submit the command again.

AMQ7016 There is an error in the maximum number of uncommitted messages specified.

Explanation: You specified a value for the maximum number of uncommitted messages to be allowed that is not valid. The value must be not less than 1 and not greater than 999 999 999.

User action: Correct the value and submit the command again.

AMQ7017 Log not available.

Explanation: The queue manager was unable to use the log. This could be due to a log file being missing or damaged, or the log path to the queue manager being inaccessible.

User action: Ensure that the LogPath attribute in the queue manager configuration file is correct. If a log file is missing or otherwise unusable, restore a backup copy of the file, or the entire queue manager.

AMQ7018 The queue manager has stopped

AMQ7019 An error occurred while creating the directory structure for the new queue manager.

Explanation: During creation of the queue manager an error occurred while trying to create a file or directory.

User action: Identify why the queue manager files cannot be created. It is probable that there is insufficient space on the specified disk, or that there is a problem with access control. Correct the problem and submit the command again.

AMQ7021 An error occurred while deleting the directory structure for the queue manager.

Explanation: While deleting the queue manager, an error occurred deleting a file or directory. The queue manager may not have been completely deleted.

User action: Follow problem determination procedures to identify the file or directory and to complete deletion of the queue manager.

AMQ7024 Arguments supplied to a command are not valid.

Explanation: You supplied arguments to a command that it could not interpret. It is probable that you specified a flag not accepted by the command, or that you included extra flags.

User action: Correct the command and submit it again.

AMQ7025 Error in the supplied command description.

Explanation: The descriptive text you supplied on the command was in error.

User action: Correct the descriptive text and submit the command again.

AMQ7026 A principal or group name was invalid.

Explanation: You specified the name of a principal or group which does not exist.

User action: Correct the name and resubmit the request.

AMQ7028 The queue manager is not available for use.

Explanation: You have requested an action that requires the queue manager running, however, the queue manager is not currently running.

User action: Start the required queue manager and submit the command again.

AMQ7030 Request to quiesce the queue manager accepted. The queue manager will stop when there is no further work for it to perform.

Explanation: You have requested that the queue manager end when there is no more work for it. In the meantime, it will refuse new applications that attempt to start, although it allows those already running to complete their work.

User action: None.

AMQ7031 The queue manager is stopping.

Explanation: You issued a command that requires the queue manager running, however, it is currently in the process of stopping. The command cannot be run.

User action: None

AMQ7041 Object already exists.

Explanation: A Define Object operation was performed, but the name selected for the object is already in use by an object that is unknown to MQSeries. The object name selected by MQSeries was &3, in directory &4, of object type &5.

User action: Remove the conflicting object from the MQSeries system, then try the operation again.

AMQ7042 Media image not available for object &3 of type &4.

Explanation: The media image for object &3, type &4, is not available for media recovery. A log file containing part of the media image cannot be accessed.

User action: A previous message indicates which log file could not be accessed. Restore a copy of the log file and all subsequent log files from backup. If this is not possible, you must delete the object instead.

AMQ7044 Media recovery not allowed.

Explanation: Media recovery is not possible on a queue manager using a circular log. Damaged objects must be deleted on such a queue manager.

User action: None.

AMQ7047 An unexpected error was encountered by a command.

Explanation: An internal error occurred during the processing of a command.

User action: Follow problem determination procedures to identify the cause of the error.

AMQ7048 The queue manager name is either not valid or not known

Explanation: Either the specified queue manager name does not conform to the rules required by MQSeries or the queue manager does not exist. The rules for naming MQSeries objects are detailed in the MQSeries Command Reference.

User action: Correct the name and submit the command again.

AMQ7053 The transaction has been committed.

Explanation: The prepared transaction has been committed.

User action: None.

AMQ7054 The transaction has been backed out.

Explanation: The prepared transaction has been backed out.

User action: None.

AMQ7055 The transaction number is not recognized.

Explanation: The number of the transaction you supplied was not recognized as belonging to an in-doubt transaction.

User action: Ensure that you entered a valid transaction number. It is possible that the transaction number you entered corresponds to a transaction which was committed or backed out before you issued the command to resolve it.

AMQ7056 Transaction number &1,&2.

Explanation: This message is used to report the number of an in-doubt transaction.

User action: None.

AMQ7064 Log path not valid or inaccessible.

Explanation: The supplied log path could not be used by the queue manager. Possible reasons for this include the path not existing, the queue manager not being able to write to the path, or the path residing on a remote device.

User action: Ensure that the log path exists and that the queue manager has authority to read and write to it. If the queue manager already exists, ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ7065 Insufficient space on disk.

Explanation: The operation cannot be completed due to shortage of disk space.

User action: Either make more disk space available, or reduce the disk requirements of the command you issued.

AMQ7066 There are no prepared transactions.

Explanation: There are no prepared transactions to be resolved.

User action: None.

AMQ7068 Authority file contains an authority stanza that is not valid.

Explanation: A syntax error has been found in one of the files containing authorization information for the queue manager.

User action: Correct the contents of the incorrect authorization file by editing it.

AMQ7069 The queue manager was created successfully, but cannot be made the default.

Explanation: The queue manager was defined to be the default queue manager for the machine when it was created. However, although the queue manager has been created, an error occurred trying to make it the default. There may not be a default queue manager defined for the machine at present.

User action: There is probably a problem with the machine-wide ini file. Verify the existence of the file, its access permissions, and its contents. If its backup file exists, reconcile the contents of the two files and then delete the backup. Finally, either update the machine-wide ini file by hand to specify the desired default queue manager, or delete and recreate the queue manager.

AMQ7073 Log size not valid.

Explanation: Either the number of log files or the size of the log files was outside the accepted values.

User action: Make sure that the log parameters you enter lie within the valid range.

AMQ7074 Unknown stanza key &4 on line &1 of ini file &3.

Explanation: Line &1 of the configuration file &3 contained a stanza called &3. This stanza is not recognized.

User action: Check the contents of the file and retry the operation.

AMQ7075 Unknown attribute &4 on line &1 of ini file &3.

Explanation: Line &1 of the configuration file &3 contained an attribute called &4 that is not valid. This attribute is not recognized in this context.

User action: Check the contents of the file and retry the operation.

AMQ7076 Value &5 not valid for attribute &4 on line &1 of ini file &3

Explanation: Line &1 of the configuration file &3 contained value &5 that is not valid for the attribute &4.

User action: Check the contents of the file and retry the operation.

AMQ7077 You are not authorized to perform the requested operation.

Explanation: You tried to issue a command for the queue manager. You are not authorized to perform the command.

User action: Contact your system administrator to perform the command for you. Alternatively, request authority to perform the command from your system administrator.

AMQ7080 No objects processed.

Explanation: No objects were processed, either because no objects matched the criteria given, or because the objects found did not require processing.

User action: None.

AMQ7081 Object &3, type &4 recreated.

Explanation: The object &3, type &4 was recreated from its media image.

User action: None.

AMQ7082 Object &3, type &4 is not damaged.

Explanation: Object &3, type &4 cannot be recreated since it is not damaged.

User action: None.

AMQ7083 A resource problem was encountered by a command.

Explanation: The command failed due to a resource problem. Possible causes include the log being full or the command running out of memory.

User action: Look at the previous messages to diagnose the problem. Rectify the problem and retry the operation.

AMQ7084 Object &3, type &4 damaged.

Explanation: The object &3, type &4 was damaged. The object must be deleted or, if the queue manager supports media recovery, recreated from its media image.

User action: Delete the object or recreate it from its media image.

AMQ7085 Object &3, type &4 not found.

Explanation: Object &3, type &4 cannot be found.

User action: None.

AMQ7086 Media image for object &3, type &4 recorded.

Explanation: The media image for object &3, type &4 has been recorded.

User action: None.

AMQ7087 Object &3, type &4 is a temporary object

Explanation: Object &3, type &4 is a temporary object. Media recovery operations are not permitted on temporary objects.

User action: None.

AMQ7088 Object &3, type &4 in use.

Explanation: Object &3, type &4 is in use. Either an application has it open or, if it is a local queue, there are uncommitted messages on it.

User action: Ensure that the object is not opened by any applications, and that there are no uncommitted messages on the object, if it is a local queue. Then, retry the operation.

AMQ7089 Media recovery already in progress.

Explanation: Another media recovery operation is already in progress. Only one media recovery operation is permitted at a time.

User action: Wait for the existing media recovery operation to complete and retry the operation.

AMQ7090 The queue manager CCSID is not valid.

Explanation: The CCSID to be used by the QMGR is not valid, probably because it is a DBCS CCSID.

User action: None.

AMQ7091 You are performing authorization for the queue manager, but you specified an object name.

Explanation: Modification of authorizations for a queue manager can be performed only from that queue manager. You must not specify an object name.

User action: Correct the command and submit it again.

AMQ7092 An object name is required but you did not specify one.

Explanation: The command needs the name of an object, but you did not specify one.

User action: Correct the command and submit it again.

AMQ7093 An object type is required but you did not specify one.

Explanation: The command needs the type of the object, but you did not specify one.

User action: Correct the command and submit it again.

AMQ7094 You specified an object type that is not valid, or more than one object type.

Explanation: Either the type of object you specified was not valid, or you specified multiple object types on a command which supports only one.

User action: Correct the command and submit it again.

AMQ7095 An entity name is required but you did not specify one.

Explanation: The command needs one or more entity names, but you did not specify any. Entities can be principals or groups.

User action: Correct the command and submit it again.

AMQ7096 An authorization specification is required but you did not provide one.

Explanation: The command sets the authorizations on MQSeries objects. However you did not specify which authorizations are to be set.

User action: Correct the command and submit it again.

AMQ7097 You gave an authorization specification that is not valid.

Explanation: The authorization specification you provided to the command contained one or more items that could not be interpreted.

User action: Correct the command and submit it again.

AMQ7098 The command accepts only one entity name. You specified more than one.

Explanation: The command can accept only one principal or group name. You specified more than one.

User action: Correct the command and submit it again.

AMQ7099 Entity &3 has the following authorizations for object &4:

Explanation: Informational message. The list of authorizations follows.

User action: None.

AMQ7305 Trigger message could not be put on an initiation queue.

Explanation: The attempt to put a trigger message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the dead-letter queue.

User action: Ensure that the initiation queue is available, and operational.

AMQ7306 The dead-letter queue must be a local queue.

Explanation: An undelivered message has not been put on the dead-letter queue &4 on queue manager &5, because the queue is not a local queue. The message will be discarded.

User action: Inform your system administrator.

AMQ7307 A message could not be put on the dead-letter queue.

Explanation: The attempt to put a message on the undelivered-message queue &4 on queue manager &5 failed with reason code &1. The message will be discarded.

User action: Ensure that the undelivered-message queue is available, and operational.

AMQ7308 Trigger condition &1 was not satisfied.

Explanation: At least one of the conditions required for generating a trigger message was not satisfied, so a trigger message was not generated. If you were expecting a trigger message, consult the MQSeries Application Programming Guide for a list of the conditions required. (Note that arranging for condition &1 to be satisfied might not be sufficient because the conditions are checked in an arbitrary order, and checking stops when the first unsatisfied condition is discovered.)

User action: If a trigger message is required, ensure that all the conditions for generating one are satisfied.

AMQ7310 Report message could not be put on a reply-to queue.

Explanation: The attempt to put a report message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the undelivered-message queue.

User action: Ensure that the reply-to queue is available, and operational.

AMQ7463 The log for queue manager &3 is full.

Explanation: This message is issued when an attempt to write a log record is rejected because the log is full. The queue manager will attempt to resolve the problem.

User action: This situation may be encountered during a period of unusually high message traffic. However, if you persistently fill the log, you may have to consider enlarging the size of the log. You can either increase the number of log files by changing the values in the queue manager configuration file. You will then have to stop and restart the queue manager. Alternatively, if you need to make the log files themselves bigger, you will have to delete and recreate the queue manager.

AMQ7464 The log for queue manager &3 is no longer full.

Explanation: This message is issued when a log was previously full, but an attempt to write a log record has now been accepted. The log full situation has been resolved.

User action: None

AMQ7465 The log for queue manager &3 is full. This is due to the presence of a long-running transaction.

Explanation: This message is issued when an attempt made to resolve a log full situation fails, because the space is occupied by a long-running transaction.

User action: Try to ensure that the duration of your transactions is not excessive. Commit or roll back any old transactions to release log space for further log records.

AMQ7466 The log for queue manager &3 is too small to support the current data rate.

Explanation: This message is issued when the monitoring tasks maintaining the log cannot keep up with the current rate of data being written.

User action: The number of primary log files configured should be increased to prevent possible log full situations.

AMQ7467 The oldest log file required to start queue manager &3 is &4.

Explanation: The log file &4 contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

User action: You can move log files older than &4 to an archive medium to release space in the log directory. If you move any of the log files required to recreate objects from their media images, you will have to restore them to recreate the objects.

AMQ7468 The oldest log file required to perform media recovery of queue manager &3 is &4.

Explanation: The log file &4 contains the oldest log record required to recreate any of the objects from their media images. Any log files prior to this will not be accessed by media recovery operations.

User action: You can move log files older than &4 to an archive medium to release space in the log directory.

AMQ7469 Transactions rolled back to release log space.

Explanation: The log space for the queue manager is becoming full. One or more long-running transactions have been rolled back to release log space so that the queue manager can continue to process requests.

User action: Try to ensure that the duration of your transactions is not excessive. You may consider increasing the size of the log to allow transactions to last longer before the log starts to become full.

AMQ7472 Object &3, type &4 damaged.

Explanation: Object &3, type &4 has been marked as damaged. This indicates that the queue manager was either unable to access the object in the file system, or that some kind of inconsistency with the data in the object was detected.

User action: If a damaged object is detected, the action performed depends on whether the queue manager supports media recovery and when the damage was detected. If the queue manager does not support media recovery, you must

delete the object as no recovery is possible. If the queue manager does support media recovery and the damage is detected during the processing performed when the queue manager is being started, the queue manager will automatically initiate media recovery of the object. If the queue manager supports media recovery and the damage is detected once the queue manager has started, it may be recovered from a media image using the rcrmqobj command or it may be deleted.

AMQ7901 The data-conversion exit &3 has not loaded.

Explanation: The data-conversion exit program, &3, failed to load. The internal function gave exception &4.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7902 The data conversion exit &3 was not loaded. The operating system call &4 returned &1.

Explanation:

User action: Specify REPLACE to over-write the existing file, or choose a different output file name.

AMQ7903 The data-conversion exit &3 cannot be found.

Explanation: Message data conversion has been requested for an MQSeries message with a user-defined format, but the necessary data-conversion exit program, &3, cannot be found. The internal function gave exception &4.

User action: Check that the necessary data-conversion exit &3 exists.

AMQ7904 The data conversion exit &3 cannot be found, or loaded.

Explanation: Message data conversion was requested for an MQSeries message with a user-defined format, but the necessary data conversion exit program, &3, was not found, or loaded. The &4 function call gave a return code of &1.

User action: Check that the necessary data conversion exit routine exists one of the standard directories for dynamically loaded modules. If necessary, inspect the generated output to examine the message descriptor (MQMD structure) of the MQSeries message for which conversion was requested. This may help you to determine where the message originated.

AMQ7905 Unexpected exception &4 in data-conversion exit.

Explanation: The data-conversion exit program, &3, ended with an unexpected exception &4. The message has not been converted.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7907 Unexpected exception in data-conversion exit.

Explanation: The data-conversion exit routine, &3, ended with an unexpected exception. The message has not been converted.

User action: Correct the error in the data-conversion exit routine.

AMQ7921 An internal MQSeries error occurred.

Explanation: The MQDXP structure passed to the Internal Formats Conversion routine contains an incorrect eyecatcher field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7922 A PCF message is incomplete.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the message is only &1 bytes long and does not contain a PCF header. The message has either been truncated, or it contains data that is not valid.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7923 A message had an unrecognized integer encoding.

Explanation: Message data conversion cannot convert a message because the integer encoding value of the message, &1, was not recognized.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7924 Bad length in the PCF header (length = &1).

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the PCF header structure contains an incorrect length field. Either the message has been truncated, or it contains data that is not valid.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7925 Message version &1 is not supported.

Explanation: Message data conversion cannot convert a message because the Version field of the message contains an incorrect value.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7926 A PCF message has an incorrect parameter count value &1.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because the parameter count field of the PCF header is incorrect.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7927 Bad type in PCF structure number &1 (type = &2).

Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect type field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7928 Bad length in PCF structure number &1 (length = &2).

Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect length field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7929 A PCF structure is incomplete.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number &1, of Type value &2, within the message is incomplete. The message has either been truncated, or it contains data that is not valid.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7930 Bad CCSID in PCF structure number &1 (CCSID = &2).

Explanation: A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contains an incorrect CCSID.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7931 Bad length in PCF structure number &1 (length = &2).

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because one of the structures of the message contains an incorrect length field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7932 Bad count in PCF structure number &1 (count = &2).

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because a StringList structure of the message contains an incorrect count field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message, and to see how data that is not valid became included in the message.

AMQ7933 Bad string length in PCF structure.

Explanation: Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number &1 of the message contains an incorrect string length value &2.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7934 Wrong combination of MQCCSI_DEFAULT with MQCCSI_EMBEDDED.

Explanation: Message data conversion could not convert a message in Programmable Command Format (PCF) because structure &1 of the message contained a CodedCharSetId field of MQCCSI_DEFAULT while the message itself had a CodedCharSetId of MQCCSI_EMBEDDED. This is an incorrect combination.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7935 Bad CCSID in message header (CCSID = &1).

Explanation: Message data conversion could not convert a message because the Message Descriptor of the message contained an incorrect CodedCharSetId field.

User action: Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7936 The file &3 already exists.

Explanation: The output file already exists, but REPLACE has not been specified.

User action: Specify REPLACE to over-write the existing file, or select a different output file name.

**AMQ7943 Usage: crtmqcvx SourceFile TargetFile
AMQ7953 One structure has been parsed.**

Explanation: The crtmqcvx command has parsed one structure.

User action: None.

AMQ7954 &1 structures have been parsed.

Explanation: The crtmqcvx command has parsed %1 structures.

User action: None.

AMQ7955 Unexpected field: &1.

Explanation: The field within the structure is of a type that is not recognized.

User action: Correct the field and retry the command.

AMQ7956 Bad array dimension.

Explanation: An array field of the structure has an incorrect dimension value.

User action: Correct the field and retry the command.

AMQ7957 Warning at line &1.

Explanation: The structure contains another field after a variable length field.

User action: Correct the structure and retry the command.

AMQ7958 Error at line &1 in field &3.

Explanation: Field name '&3' is a field of type 'float'. Fields of type float are not supported by this command.

User action: Either correct the structure to eliminate fields of type float, or write your own routine to support conversion of these fields.

AMQ7959 Error at line &1 in field &3.

Explanation: Field name '&3' is a field of type 'double'. Fields of type double are not supported by this command.

User action: Either correct the structure to eliminate fields of type double, or write your own routine to support conversion of these fields.

AMQ7960 Error at line &1 in field &3.

Explanation: Field name '&3' is a 'pointer' field. Fields of type pointer are not supported by this command.

User action: Either correct the structure to eliminate fields of type pointer, or write your own routine to support conversion of these fields.

AMQ7961 Error at line &1 in field &3.

Explanation: Field name '&3' is a 'bit' field. Bit fields are not supported by this command.

User action: Either correct the structure to eliminate bit fields, or write your own routine to support conversion of these fields.

AMQ7962 No input file specified.

Explanation: This command requires that an input file is specified.

User action: Specify the name of the input file and retry the command.

AMQ7963 No output file specified.

Explanation: This command requires that an output file name is specified.

User action: Specify the name of the output file and retry the command.

AMQ7964 Unexpected option &3.

Explanation: The option specified is not valid for this command.

User action: Retry the command with a valid option.

AMQ7965 Incorrect number of arguments.

Explanation: The command was passed an incorrect number of arguments.

User action: Retry the command, passing it the correct number of arguments.

AMQ7968 Cannot open file '&3'.

Explanation: You cannot open the file &3.

User action: Check that you have the correct authorization to the file and retry the command.

AMQ7969 Syntax error.

Explanation: This line of the input file contains a language syntax error.

User action: Correct the syntax error and retry the command.

AMQ7970 Syntax error on line &1.

Explanation: This message identifies where, in the input file, a previously reported error was detected.

User action: Correct the error and retry the command.

Administration messages

AMQ8001 MQSeries queue manager created.

Explanation: MQSeries queue manager &5 created.
User action: None.

AMQ8002 MQSeries queue manager deleted.

Explanation: MQSeries queue manager &5 deleted.
User action: None.

AMQ8003 MQSeries queue manager started.

Explanation: MQSeries queue manager &5 started.
User action: None.

AMQ8004 MQSeries queue manager ended.

Explanation: MQSeries queue manager &5 ended.
User action: None.

AMQ8005 MQSeries queue manager changed.

Explanation: MQSeries queue manager &5 changed.
User action: None.

AMQ8006 MQSeries queue created.

Explanation: MQSeries queue &5 created.
User action: None.

AMQ8007 MQSeries queue deleted.

Explanation: MQSeries queue &5 deleted.
User action: None.

AMQ8008 MQSeries queue changed.

Explanation: MQSeries queue &5 changed.
User action: None.

AMQ8010 MQSeries process created.

Explanation: MQSeries process &5 created.
User action: None.

AMQ8011 MQSeries process deleted.

Explanation: MQSeries process &5 deleted.
User action: None.

AMQ8012 MQSeries process changed.

Explanation: MQSeries process &5 changed.
User action: None.

AMQ8013 MQM process copied.

Explanation: MQM process &5 created in library &3 by copying.
User action: None.

AMQ8014 MQSeries channel created.

Explanation: MQSeries channel &5 created.
User action: None.

AMQ8015 MQSeries channel deleted.

Explanation: MQSeries channel &5 deleted.
User action: None.

AMQ8016 MQSeries channel changed.

Explanation: MQSeries channel &5 changed.
User action: None.

AMQ8018 Start MQSeries channel accepted.

Explanation: MQSeries channel &5 is being started. The start channel function has been initiated. This involves a series of operations across the network before the channel is actually started. The channel status displays "BINDING" for a short period while communication protocols are negotiated with the channel with whom communication is being initiated.
User action: None.

AMQ8019 Stop MQSeries channel accepted.

Explanation: MQSeries channel &5 has been requested to stop.
User action: None.

AMQ8020 Ping MQSeries channel complete.

Explanation: Ping MQSeries channel &5 complete.
User action: None.

AMQ8021 MQSeries Listener program started.

Explanation: The MQSeries channel listener program has been started.
User action: None.

AMQ8022 MQSeries queue cleared.

Explanation: All messages on MQSeries queue &5 have been deleted.
User action: None.

AMQ8023 MQSeries channel reset.

Explanation: MQSeries channel &5 has been reset.
User action: None.

AMQ8024 MQSeries channel initiator started.

Explanation: The channel initiator for MQSeries queue &5 has been started.
User action: None.

AMQ8025 MQSeries channel resolved.

Explanation: In doubt messages for MQSeries channel &5 have been resolved.
User action: None.

AMQ8026 End MQSeries queue manager accepted.

Explanation: A controlled stop request has been initiated for MQSeries queue manager &5.
User action: None.

AMQ8027 MQSeries command server started.

Explanation: The MQSeries command server has been started.
User action: None.

AMQ8028 MQSeries command server ended.

Explanation: The MQSeries command server has been stopped.
User action: None.

AMQ8029 MQSeries authority granted.**Explanation:** Authority for MQSeries object &5 granted.**User action:** None.**AMQ8030 MQSeries authority revoked.****Explanation:** Authority for MQSeries object &5 revoked.**User action:** None.**AMQ8033 MQSeries object recreated.****Explanation:** MQSeries object &5 has been recreated from image.**User action:** None.**AMQ8034 MQSeries object image recorded.****Explanation:** Image of MQSeries object &5 has been recorded.**User action:** None.**AMQ8035 MQSeries Command Server Status . . . :
Running****AMQ8036 MQSeries command server status . . . :
Stopping****AMQ8037 MQSeries command server status . . . :
Starting****AMQ8038 MQSeries command server status . . . :
Running with queue disabled****AMQ8039 MQSeries command server status . . . :
Stopped****AMQ8040 MQSeries command server ending.****AMQ8041 The queue manager cannot be restarted because processes, that were previously connected, are still running.****Explanation:** Processes, that were connected to the queue manager the last time it was running, are still active. The queue manager cannot be restarted.**User action:** Stop the processes and try to start the queue manager.**AMQ8042 Process &1 is still running.****AMQ8043 Non runtime application attempted to connect to runtime only queue manager.****Explanation:** A non runtime application attempted to connect to a queue manager on a node where support for non runtime applications has not been installed. The connect attempt will be rejected with a reason of MQRC_ENVIRONMENT_ERROR.**User action:** If the node is intended to support only runtime applications then investigate why a non runtime application has attempted to connect to the queue manager. If the node is intended to support non runtime only applications then investigate if the base option has been installed. The base option must be installed if non runtime applications are to run on this node.**AMQ8101 Unexpected error (&1).****Explanation:** An unexpected reason code with hexadecimal value &4 was received from the MQSeries queue manager during command processing. (Note that hexadecimal values in the range X'07D1'-X'0BB7' correspond to MQI reason codes 2001-2999.) More information might be available in the log. If the reason code value indicates that the error was associated with a particular parameter, the parameter concerned is &2.**User action:** Correct the error and then try the command again.**AMQ8102 MQSeries object name specified in &2 not valid.****Explanation:** MQSeries object name &5 specified in &2 is not valid. The length of the name must not exceed 48 characters, or 20 characters if it is a channel name. The name should contain the following characters only: lowercase a-z, uppercase A-Z, numeric 0-9, period (.), forward slash (/), underscore (_) and percent sign (%).**User action:** Change the length of the parameter value or change the parameter value to contain a valid combination of characters, then try the command again.**AMQ8103 Insufficient storage available.****Explanation:** There was insufficient storage available to perform the requested operation.**User action:** Free some storage and then try the command again.**AMQ8104 MQSeries directory &3 not found.****Explanation:** Directory &3 was not found. This directory is created when MQSeries is installed successfully. Refer to the log for more information.**User action:** Verify that installation of MQSeries was successful. Correct the error and then try the command again.**AMQ8105 Object error.****Explanation:** An object error occurred. Refer to the log for more information.**User action:** Correct the error and then try the command again.**AMQ8106 MQSeries queue manager being created.****Explanation:** The MQSeries queue manager is being created.**User action:** Wait for the creation process to complete and then try the command again.**AMQ8107 MQSeries queue manager running.****Explanation:** The MQSeries queue manager is running.**User action:** None.**AMQ8108 MQSeries queue manager ending.****Explanation:** The MQSeries queue manager is ending.**User action:** Wait for the MQSeries queue manager to end and then try the command again.**AMQ8109 MQSeries queue manager being deleted.****Explanation:** The MQSeries queue manager is being deleted.**User action:** Wait for the deletion process to complete.**AMQ8110 MQSeries queue manager already exists.****Explanation:** MQSeries queue manager &5 already exists.**User action:** None.

AMQ8117 MQSeries queue manager deletion incomplete.

Explanation: Deletion of MQSeries queue manager &5 was only partially successful. An object was not found, or could not be deleted. Refer to the log for more information.

User action: Delete any remaining MQSeries queue manager objects.

AMQ8118 MQSeries queue manager does not exist.

Explanation: MQSeries queue manager &5 does not exist.

User action: Create the message queue manager (crtmqm command) and then try the command again.

AMQ8135 Not authorized.

Explanation: You are not authorized to perform the requested operation for the MQSeries object &5 specified in &2. Either you are not authorized to perform the requested operation, or you are not authorized to the specified MQSeries object. For a copy command, you may not be authorized to the specified source MQSeries object, or, for a create command, you may not be authorized to the system default MQSeries object of the specified type.

User action: Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8137 MQSeries queue manager already starting.

Explanation: The strmqm command was unsuccessful because MQSeries queue manager &5 is already starting.

User action: Wait for the strmqm command to complete.

AMQ8138 The MQSeries queue has an incorrect type.

Explanation: The operation is not valid with MQSeries queue &5 because it is not a local queue.

User action: Change the QNAME parameter to specify an MQSeries queue of the correct type.

AMQ8139 Already connected.

Explanation: A connection to the MQSeries queue manager already exists.

User action: None.

AMQ8140 Resource timeout error.

Explanation: A timeout occurred in the communication between internal MQSeries queue manager components. This is most likely to occur when the system is heavily loaded.

User action: Wait until the system is less heavily loaded, then try the command again.

AMQ8141 MQSeries queue manager starting.

Explanation: MQSeries queue manager &5 is starting.

User action: Wait for the MQSeries queue manager startup process to complete and then try the command again.

AMQ8142 MQSeries queue manager stopped.

Explanation: MQSeries queue manager &5 is stopped.

User action: Use the strmqm command to start the MQSeries queue manager, and then try the command again.

AMQ8143 MQSeries queue not empty.

Explanation: MQSeries queue &5 specified in &2 is not empty or contains uncommitted updates.

User action: Commit or rollback any uncommitted updates. If the command is DELETE QLOCAL, use the CLEAR QLOCAL command to clear the messages from the MQSeries queue. Then try the command again.

AMQ8144 Log not available.

Explanation: The MQSeries logging resource is not available.

User action: Use the dltmqm command to delete the MQSeries queue manager and then the crtmqm command to create the MQSeries queue manager. Then try the command again.

AMQ8145 Connection broken.

Explanation: The connection to the MQSeries queue manager failed during command processing. This may be caused by an endmqm -i command being issued by another user, or by an MQSeries queue manager error.

User action: Use the strmqm command to start the message queue manager, wait until the message queue manager has started, and try the command again.

AMQ8146 MQSeries queue manager not available.

Explanation: The MQSeries queue manager is not available because it has been stopped or has not been created.

User action: Use the crtmqm command to create the message queue manager, or the strmqm command to start the message queue manager as necessary. Then try the command again.

AMQ8147 MQSeries object not found.

Explanation: If the command entered was Change, the MQSeries object &5 specified in &2 does not exist. If the command entered was Copy, the source MQSeries object does not exist. If the command entered was Create, the system default MQSeries object of the specified type does not exist.

User action: Correct the MQSeries object name and then try the command again or, if you are creating a new MQSeries queue or process object, either specify all parameters explicitly or ensure that the system default object of the required type exists. The system default queue names are SYSTEM.DEFAULT.LOCAL.QUEUE, SYSTEM.DEFAULT.ALIAS.QUEUE and SYSTEM.DEFAULT.REMOTE.QUEUE. The system default process name is SYSTEM.DEFAULT.PROCESS.

AMQ8148 MQSeries object in use.

Explanation: MQSeries object &5 specified in &2 is in use by an MQSeries application program.

User action: Wait until the MQSeries object is no longer in use and then try the command again, or specify FORCE to force the processing of the MQSeries ALTER command regardless of any application program affected by the change. If the object is the dead-letter queue and the open input count is nonzero, it may be in use by an MQSeries channel. If the object is another MQSeries queue object with a nonzero open output count, it may be in use by an MQSeries channel (of type RCVR or RQSTR). In either case, use the STOP CHANNEL and START CHANNEL commands to stop and restart the channel in order to solve the problem.

AMQ8149 MQSeries object damaged.

Explanation: The MQSeries object &5 specified in &2 is damaged.

User action: The MQSeries object contents are not valid. Issue the DISPLAY CHANNEL, DISPLAY QUEUE, or DISPLAY PROCESS command, as required, to determine the name of the damaged object. Issue the DEFINE command, for the appropriate object type, to replace the damaged object, then try the command again.

AMQ8150 MQSeries object already exists.

Explanation: MQSeries object &5 specified for &2 could not be created because it already exists.

User action: Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries object. Then try the command again.

AMQ8151 MQSeries object has different type.

Explanation: The type specified for MQSeries object &5 is different from the type of the object being altered or defined.

User action: Use the correct MQSeries command for the object type, and then try the command again.

AMQ8152 Source MQSeries object has different type.

Explanation: The type of the source MQSeries object is different from that specified.

User action: Correct the name of the command, or source MQSeries object name, and then try the command again, or try the command using the REPLACE option.

AMQ8153 Insufficient disk space for the specified queue.

Explanation: The command failed because there was insufficient disk space available for the specified queue.

User action: Release some disk space and then try the command again.

AMQ8155 Connection limit exceeded.

Explanation: The queue manager connection limit has been exceeded.

User action: The maximum limit on the number of MQSeries application programs that may be connected to the MQSeries queue manager has been exceeded. Try the command later.

AMQ8156 MQSeries queue manager quiescing.

Explanation: The MQSeries queue manager is quiescing.

User action: The queue manager was stopping with -c specified for endmqm. Wait until the queue manager has been restarted and then try the command again.

AMQ8157 Security error.

Explanation: An error was reported by the security manager program.

User action: Inform your systems administrator, wait until the problem has been corrected, and then try the command again.

AMQ8159 MAXDEPTH not allowed with queue type *ALS or *RMT.

Explanation: The MAXDEPTH parameter may not be specified for an MQM queue of type *ALS or *RMT.

User action: Remove the MAXDEPTH parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8160 DFTSHARE not allowed with queue type *ALS or *RMT.

Explanation: The DFTSHARE parameter may not be specified for an MQM queue of type *ALS or *RMT.

User action: Remove the DFTSHARE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8172 Already disconnected.

Explanation: The MQI reason code of 2018 was returned from the MQSeries queue manager in response to an MQDISC request issued during command processing.

User action: None.

AMQ8173 No processes to display.

Explanation: There are no matching processes defined on this system.

User action: Using the DEFINE PROCESS command to create a process.

AMQ8174 No queues to display.

Explanation: There are no matching queues defined on this system.

User action: Using the appropriate command to define a queue of the type that you require, that is, DEFINE QALIAS, DEFINE QLOCAL, DEFINE QMODEL, or DEFINE QREMOTE.

AMQ8185 Operating system object already exists.

Explanation: The MQSeries object cannot be created because an object that is not known to MQSeries already exists in the MQSeries directory with the name that should be used for the new object. Refer to the log for previous messages.

User action: Remove the non-MQSeries object from the MQSeries library, and try the command again.

AMQ8186 Image not available for MQSeries object &5.

Explanation: MQSeries object &5 type &3 cannot be recreated because the image is not fully available in the logs that are currently online. Refer to earlier messages in the error log for information about the error logs that need to be brought online for this object to be recreated.

User action: Bring the relevant error logs online, and try the command again.

AMQ8187 MQSeries object &5 is currently open.

Explanation: MQSeries object &5, type &3, is currently in use, so the &1 command cannot be issued against it. If a generic list was presented to the command, the command is still issued against the other objects in the list.

User action: Wait until the object is no longer in use, and try the command again.

AMQ8188 Insufficient authorization to MQSeries object &5.

Explanation: You are not authorized to issue the &1 command against MQSeries object &5 type &3. If a generic list was presented to the command, the command is still issued against the other objects in the list.

User action: Obtain sufficient authorization for the object, and retry the command.

AMQ8189 MQSeries object &5 is damaged.

Explanation: MQSeries object &5 type &3 is damaged and the &1 command cannot be issued against it. If a generic list was presented to the command then the command is still issued against the other objects in the list.

User action: Issue the appropriate DEFINE command for the object, specifying REPLACE, and then try the command again.

AMQ8190 &1 succeeded on &2 objects and failed on &3 objects.

Explanation: An operation performed on a generic list of objects was not completely successful.

User action: Examine the log for details of the errors encountered, and take appropriate action.

AMQ8191 MQSeries command server is starting.

Explanation: The MQSeries command server is starting.

User action: Wait for the strmqcsv command to complete and then try the operation again.

AMQ8192 MQSeries command server already starting.

Explanation: The request to start the MQSeries command server was unsuccessful because the MQSeries command server is already starting.

User action: Wait for the strmqcsv command to complete.

AMQ8193 MQSeries command server is ending.

Explanation: The MQSeries command server is ending.

User action: Wait for the endmqcsv command to complete and then try the command again.

AMQ8194 MQSeries command server already ending.

Explanation: The end MQSeries command server request was unsuccessful because the MQSeries command server is already ending.

User action: Wait for the endmqcsv command to complete.

AMQ8195 MQSeries command server already running.

Explanation: The strmqcsv command was unsuccessful because the MQSeries command server is already running.

User action: None.

AMQ8196 MQSeries command server already stopped.

Explanation: The request to end the MQSeries command server was unsuccessful because the MQSeries command server is already stopped.

User action: None.

AMQ8197 Deleted MQSeries queue damaged.

Explanation: The deleted MQSeries queue &5 was damaged, and any messages it contained have been lost.

User action: None.

AMQ8226 MQSeries channel already exists.

Explanation: MQSeries channel &3 cannot be created because it already exists.

User action: Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries channel and then try the command again.

AMQ8227 Channel &3 not found.

Explanation: ALTER CHANNEL has been issued for a non-existent channel.

User action: Correct the MQSeries channel name and then try the command again.

AMQ8296 &4 MQSC commands completed successfully.

Explanation: The &1 command has completed successfully. The &4 MQSeries commands from &5 have been processed without error and a report written to the printer spool file.

User action: None.

AMQ8297 &4 MQSC commands verified successfully.

Explanation: The &1 command completed successfully.

The &4 MQSeries commands from &5 have been verified and a report written to the printer spool file.

User action: None.

AMQ8298 Error report generated for MQSC command process.

Explanation: The &1 command attempted to process the sequence of MQSeries commands from &5 and encountered some errors, however, the operation may have partially completed. A report has been written to the printer spool file.

User action: Examine the spooled printer file for details of the errors encountered, correct the MQSC source file, and retry the operation.

AMQ8299 Cannot open &5 for MQSC process.

Explanation: The &1 command failed to open &5 for MQSeries command processing.

User action: Check that the intended file exists, and has been specified correctly. Correct the specification or create the object, and try the operation again.

AMQ8302 Internal failure initializing MQSeries services.

Explanation: An error occurred while attempting to initialize MQSeries services.

AMQ8303 Insufficient storage available to process request.**AMQ8304 Tracing cannot be started. Too many traces are already running.**

Explanation:

User action: Stop one or more of the other traces and try the command again.

AMQ8401 &1 MQSC commands read.

Explanation: The MQSC script contains &1 commands.

User action: None.

AMQ8402 &1 commands have a syntax error.

Explanation: The MQSC script contains &1 commands having a syntax error.

User action: None.

AMQ8403 &1 commands cannot be processed.

Explanation: The MQSC script contains &1 commands that failed to process.

User action: None.

AMQ8404 Command failed.

Explanation: An MQSC command has been recognized, but cannot be processed.

User action: None.

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

Explanation: The MQSC script contains &1 commands having a syntax error.

User action: None.

AMQ8406 Unexpected 'end of input' in MQSC.

Explanation: An MQSC command contains a continuation character, but the 'end of input' has been reached without completing the command.

User action: None.

AMQ8407 Display Process details.

Explanation: The MQSC DISPLAY PROCESS command completed successfully, and details follow this message.

User action: None.

AMQ8408 Display Queue Manager details.

Explanation: The MQSC DISPLAY QMGR command completed successfully, and details follow this message.

User action: None.

AMQ8409 Display Queue details.

Explanation: The MQSC DISPLAY QUEUE command completed successfully, and details follow this message.

User action: None.

AMQ8410 Parser error.

Explanation: The MQSC Parser has an internal error.

User action: None.

AMQ8411 Duplicate Keyword error.

Explanation: A command in the MQSC script contains duplicate keywords.

User action: None.

AMQ8412 Numeric Range error.

Explanation: The value assigned to an MQSC command keyword is out of the permitted range.

User action: None.

AMQ8413 String Length Error.

Explanation: A string assigned to an MQSC keyword is either NULL, or longer than the maximum permitted for that keyword.

User action: None.

AMQ8414 Display Channel details.

Explanation: The MQSC DISPLAY CHL command completed successfully, and details follow this message.

User action: None.

AMQ8415 MQSeries commands are active.

Explanation: The MQSC DISPLAY QMGR command completed successfully, and details follow this message.

User action: None.

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

Explanation: MQSC did not receive a response message from the remote command server in the time specified.

User action: None.

AMQ8417 Display Channel Status details.

Explanation: The MQSC DISPLAY CHANNEL STATUS command completed successfully, and details follow this message.

User action: None.

AMQ8418 &1 command responses received.

Explanation: Running in queued mode, &1 command responses were received from the remote command server.

User action: None.

AMQ8419 The Queue is already in the DCE cell.

Explanation: The Queue is already in the cell, that is, its SCOPE attribute is already CELL.

User action: None.

AMQ8420 Channel Status not found.

Explanation: No status was found for the specified channel(s).

User action: None.

AMQ8421 A required keyword was not specified.

Explanation: A keyword required in this command was not specified.

User action: None.

AMQ8424 Error detected in a name keyword.

Explanation: A keyword in an MQSC command contained a name string which was not valid. This may be because it contained characters which are not accepted in MQ names. Typical keywords which can produce this error are QLOCAL (and the other q types), CHANNEL, XMITQ, INITQ, MCANAME etc.

User action: None.

AMQ8498 Starting MQSeries Commands.

Explanation: The MQSC script contains &1 commands.

User action: None.

AMQ8499 Usage: runmqsc -e“ -v“ -w WaitTime“ -x“ QMgrName

Explanation: None.

User action: None.

AMQ8500 MQSeries Display MQ Files

Explanation: Title for the dspmqfls command.

User action: None.

AMQ8501 Common services initialization failed with return code &1.

Explanation: A request by the command server to initialize common services failed with return code &1.
User action: None.

AMQ8502 Connect shared memory failed with return code &1.

Explanation: A request by the command server to connect shared memory failed with return code &1.
User action: None.

AMQ8503 Post event semaphore failed with return code &1.

Explanation: A request by the command server to post an event semaphore failed with return code &1.
User action: None.

AMQ8504 Command server MQINQ failed with reason code &1.

Explanation: An MQINQ request by the command server, for the MQSeries queue &3, failed with reason code &1.
User action: None.

AMQ8505 Reallocate memory failed with return code &1.

Explanation: A request by the command server to reallocate memory failed with return code &1.
User action: None.

AMQ8506 Command server MQGET failed with reason code &1.

Explanation: An MQGET request by the command server, for the MQSeries queue &3, failed with reason code &1.
User action: None.

AMQ8507 Command server MQPUT1 request for an undelivered message failed with reason code &1.

Explanation: An attempt by the command server to put a message to the dead-letter queue, using MQPUT1, failed with reason code &1. The MQDLH reason code was &2.
User action: None.

AMQ8508 Queue Manager Delete Object List failed with return code &1.

Explanation: A request by the command server to delete a queue manager object list failed with return code &1.
User action: None.

AMQ8509 Command server MQCLOSE reply-to queue failed with reason code &1.

Explanation: An MQCLOSE request by the command server for the reply-to queue failed with reason code &1.
User action: None.

AMQ8511 Usage: strmqcsv QMgrName

AMQ8512 Usage: endmqcsv [-c | -i] QMgrName

AMQ8513 Usage: dspmqcsv QMgrName

AMQ8514 No response received after &1 seconds.

Explanation: The command server has not reported the status of running, to the start request, before the timeout of &1 seconds was reached.

User action: None.

AMQ8601 MQSeries trigger monitor started.

Explanation: The MQSeries trigger monitor has been started.

User action: None.

AMQ8602 MQSeries trigger monitor ended.

Explanation: The MQSeries trigger monitor has ended.

User action: None.

AMQ8603 Usage: runmqtrm [-m QMgrName] [-q InitQ]

AMQ8604 Use of MQSeries trigger monitor not authorized.

Explanation: The MQSeries trigger monitor cannot be run due to lack of authority to the requested queue manager or initiation queue.

User action: Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8605 Queue manager not available to the MQSeries trigger monitor

Explanation: The queue manager specified for the trigger monitor does not exist, or is not active.

User action: Check that you named the correct queue manager. Ask your systems administrator to start it, if it is not active. Then try the command again.

AMQ8606 Insufficient storage available for the MQSeries trigger monitor.

Explanation: There was insufficient storage available for the MQSeries trigger monitor to run.

User action: Free some storage and then try the command again.

AMQ8607 MQSeries trigger monitor connection failed.

Explanation: The trigger monitor's connection to the requested queue manager failed because of MQI reason code &1 from MQCONN.

User action: Consult your systems administrator about the state of the queue manager.

AMQ8608 MQSeries trigger monitor connection broken.

Explanation: The connection to the queue manager failed while the trigger monitor was running. This may be caused by an endmqm command being issued by another user, or by an MQSeries queue manager error.

User action: Consult your systems administrator about the state of the queue manager.

AMQ8609 Initiation queue missing or wrong type

Explanation: The named initiation queue could not be found; or the queue type is not correct for an initiation queue.
User action: Check that the named queue exists, and is a local queue, or that the named queue is an alias for a local queue which exists.

AMQ8610 Initiation queue in use

Explanation: The MQSeries trigger monitor could not open the initiation queue because the queue is open for exclusive use by another application.
User action: Wait until the queue is no longer in use, and try the command again.

AMQ8611 Initiation queue could not be opened.

Explanation: The MQSeries trigger monitor could not open the initiation queue; reason code &1 was returned from MQOPEN.
User action: Consult your systems administrator.

AMQ8612 Waiting for a trigger message

Explanation: The MQSeries trigger monitor is waiting for a message to arrive on the initiation queue.
User action: None.

AMQ8613 Initiation queue changed or deleted

Explanation: The MQSeries trigger monitor is unable to continue because the initiation queue has been deleted or changed since it was opened.
User action: Retry the command.

AMQ8614 Initiation queue not enabled for input.

Explanation: The MQSeries trigger monitor cannot read from the initiation queue because input is not enabled.
User action: Ask your systems administrator to enable the queue for input.

AMQ8615 MQSeries trigger monitor failed to get message.

Explanation: The MQSeries trigger monitor failed because of MQI reason code &1 from MQGET.
User action: Consult your systems administrator.

AMQ8616 End of application trigger.

Explanation: The action to trigger an application has been completed.
User action: None.

AMQ8617 Not a valid trigger message.

Explanation: The MQSeries trigger monitor received a message that is not recognized.
User action: Consult your systems administrator.

AMQ8618 Error starting triggered application.

Explanation: An error was detected when trying to start the application identified in a trigger message.
User action: Check that the application the trigger monitor was trying to start is available.

AMQ8619 Application type &1 not supported.

Explanation: A trigger message was received which specifies application type &1; the trigger monitor does not support this type.
User action: Use an alternative trigger monitor for this initiation queue.

AMQ8620 Trigger message with warning &1

Explanation: The trigger monitor received a message with a warning. For example, it may have been truncated or it could not be converted to the trigger monitor's data representation. The reason code for the warning is &1.
User action: None.

AMQ8621 Usage: runmqtmc [-m QMgrName] [-q InitQ]**AMQ8701 Usage: rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]****AMQ8702 Usage: rcrmobj [-z] [-m QMgrName] -t ObjType [GenericObjName]****AMQ8703 Usage: dspmqfls [-m QMgrName] [-t ObjType] GenericObjName****AMQ8708 Dead letter queue handler started to process INPUTQ(&3).**

Explanation: The dead letter queue handler (runmqdlq) has been started and has parsed the input file without detecting any errors and is about to start processing the queue identified in the message.
User action: None.

AMQ8709 Dead letter queue handler ending.

Explanation: The dead letter queue handler (runmqdlq) is ending because the WAIT interval has expired and there are no messages on the dead letter queue, or because the queue manager is shutting down, or because the dead letter queue handler has detected an error. If the dead letter queue handler has detected an error, an earlier message will have identified the error.
User action: None.

AMQ8721 Dead letter queue message not prefixed by a valid MQDLH.

Explanation: The dead letter queue handler (runmqdlq) retrieved a message from the nominated dead letter queue, but the message was not prefixed by a recognizable MQDLH. This typically occurs because an application is writing directly to the dead letter queue but is not prefixing messages with a valid MQDLH. The message is left on the dead letter queue and the dead letter queue handler continues to process the dead letter queue. Each time the dead letter queue handler repositions itself to a position before this message to process messages that could not be processed on a previous scan it will reprocess the failing message and will consequently reissue this message.
User action: Remove the invalid message from the dead letter queue. Do not write messages to the dead letter queue unless they have been prefixed by a valid MQDLH. If you require a dead letter queue handler that can process messages not prefixed by a valid MQDLH, you must change the sample program called amqsdlq to cater for your needs.

AMQ8722 Dead letter queue handler unable to put message: Rule &1 Reason &2.

Explanation: This message is produced by the dead letter queue handler when it is requested to redirect a message to another queue but is unable to do so. If the reason that the redirect fails is the same as the reason the message was put to the dead letter queue then it is assumed that no new error has occurred and no message is produced. The retry count for the message will be incremented and the dead letter queue handler will continue.

User action: Investigate why the dead letter queue handler was unable to put the message to the dead letter queue. The line number of the rule used to determine the action for the message should be used to help identify to which queue the dead letter queue handler attempted to PUT the message.

AMQ8741 Unable to connect to queue manager(&3) : CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not connect to the requested queue manager. This message is typically issued when the requested queue manager has not been started or is quiescing, or if the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8742 Unable to open queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the queue manager object. This message is typically issued because of a resource shortage or because the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8743 Unable to inquire on queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not inquire on the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8744 Unable to close queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not close the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8745 Unable to open dead letter queue(&3) for browse: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing. This message is typically issued because another process has opened the dead letter queue for exclusive access, or because an invalid dead letter queue name was specified. Other possible reasons include resource shortages or insufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8746 Unable to close dead letter queue: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not close the dead letter queue. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8747 Integer parameter(&2) outside permissible range for &3 on line &1.

Explanation: An integer supplied as input to the dead letter handler was outside of the valid range of values for a particular keyword.

User action: Correct the input data and restart the dead letter queue handler.

AMQ8748 Unable to get message from dead letter queue: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue. This message is typically issued because of the queue manager ending, a resource problem, or another process having deleted the dead letter queue. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8749 Unable to commit/backout action on dead letter queue: CompCode = &1 Reason

Explanation: The dead letter queue handler (runmqdlq) was unable to commit or backout an update to the dead letter queue. This message is typically issued because of the queue manager ending, or because of a resource shortage. If the queue manager has ended, the update to the dead letter queue (and any associated updates) will be backed out when the queue manager restarts. If the problem was due to a resource problem then the updates will be backed out when the dead letter queue handler terminates. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Take appropriate action based upon the completion code and reason.

AMQ8750 No valid input provided to runmqdlq.

Explanation: Either no input was provided to runmqdlq, or the input to runmqdlq contained no valid message templates. If input was provided to runmqdlq but was found to be invalid, earlier messages will have been produced explaining the cause of the error. The dead letter queue handler will end.

User action: Correct the input data and restart the dead letter queue handler.

AMQ8751 Unable to obtain private storage.

Explanation: The dead letter queue handler (runmqdlq) was unable to obtain private storage. This problem would typically arise as a result of some more global problem. For example if there is a persistent problem that is causing messages to be written to the DLQ and the same problem (for example queue full) is preventing the dead letter queue handler from taking the requested action with the message, it is necessary for the dead letter queue handler to maintain a large amount of state data to remember the retry counts associated with each message, or if the dead letter queue contains a large number of messages and the rules table has directed the dead letter queue handler to ignore the messages.

User action: Investigate if some more global problem exists, and if the dead letter queue contains a large number of messages. If the problem persists contact your support center.

AMQ8752 Parameter(&3) exceeds maximum length on line &1.

Explanation: A parameter supplied as input to the dead letter handler exceeded the maximum length for parameters of that type.

User action: Correct the input data and restart the dead letter queue handler.

AMQ8753 Duplicate parameter(&3) found on line &1.

Explanation: Two or more parameters of the same type were supplied on a single input line to the dead letter queue handler.

User action: Correct the input and restart the dead letter queue handler.

AMQ8756 Error detected releasing private storage.

Explanation: The dead letter queue handler (runmqdlq) was informed of an error while attempting to release an area of private storage. The dead letter queue handler ends.

User action: This message should be preceded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ8757 Integer parameter(&3) outside permissible range on line &1.

Explanation: An integer supplied as input to the dead letter handler was outside of the valid range of integers supported by the dead letter queue handler.

User action: Correct the input data and restart the dead letter queue handler.

AMQ8758 &1 errors detected in input to runmqdlq.

Explanation: One or more errors have been detected in the input to the dead letter queue handler(runmqdlq). Error messages will have been generated for each of these errors. The dead letter queue handler ends.

User action: Correct the input data and restart the dead letter queue handler.

AMQ8759 Invalid combination of parameters to dead letter queue handler on line &1.

Explanation: An invalid combination of input parameters has been supplied to the dead letter queue handler. Possible causes are:

no ACTION specified,
ACTION(FWD) but no FWDQ specified,
HEADER(YES|NO) specified without ACTION(FWD).

User action: Correct the input data and restart the dead letter queue handler.

AMQ8760 Unexpected failure while initializing process: Reason = &1.

Explanation: The dead letter queue handler (runmqdlq) could not perform basic initialization required to use MQ services because of an unforeseen error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8761 Unexpected failure while connecting to queue manager: CompCode = &1 Reason

Explanation: The dead letter queue handler (runmqdlq) could not connect to the requested queue manager because of an unforeseen error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8762 Unexpected error while attempting to open queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8763 Unexpected error while inquiring on queue manager: CompCode = &1 Reason = &

Explanation: The dead letter queue handler (runmqdlq) could not inquire on the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8764 Unexpected error while attempting to close queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not close the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8765 Unexpected failure while opening dead letter queue for browse: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8766 Unexpected error while closing dead letter queue: CompCode = &1 Reason = &2

Explanation: The dead letter queue handler (runmqdlq) could not close the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8767 Unexpected error while getting message from dead letter queue: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8768 Unexpected error committing/backing out action on dead letter queue: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) was unable to either commit or backout an update to the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8769 Unable to disconnect from queue manager: CompCode = &1 Reason = &2.

Explanation: The dead letter queue handler (runmqdlq) was unable to disconnect from the queue manager because of an unexpected error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action: Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

Remote messages

AMQ9001 Channel program ended normally.

Explanation: Channel program '&3' ended normally.

User action: None.

AMQ9002 Channel program started.

Explanation: Channel program '&3' started.

User action: None.

AMQ9181 The response set by the exit is not valid.

Explanation: The user exit '&3' returned a response code '&1' that is not valid in the ExitResponse field of the channel exit parameters (MQCXP). Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set a response code that is not valid.

AMQ9182 The secondary response set by the exit is not valid.

Explanation: The user exit '&3' returned a secondary response code '&1' in the ExitResponse2 field of the channel exit parameters (MQCXP) that is not valid. Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set a secondary response code that is not valid.

AMQ9184 The exit buffer address set by the exit is not valid.

Explanation: The user exit '&3' returned an address '&1' for the exit buffer that is not valid, when the secondary response code in the ExitResponse2 field of the channel exit parameters (MQCXP) is set to MQXR2_USE_EXIT_BUFFER. Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set an exit buffer address that is not valid. The most likely cause is the failure to set a value, so that the value is 0.

AMQ9189 The data length set by the exit is not valid.

Explanation: The user exit '&3' returned a data length value '&1' that was not greater than zero. Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set a data length that is not valid.

AMQ9190 Channel stopping because of an error in the exit.

Explanation: The user exit '&3', invoked for channel '&4' with id '&1' and reason '&2', returned values that are not valid, as reported in the preceding messages. The channel stops.

User action: Investigate why the user exit program set values that are not valid.

AMQ9196 Data length is larger than the agent buffer length.

Explanation: The data length '&1' set by exit '&3' is larger than the agent buffer length. The user exit returned data in the supplied agent buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set a data length that is not valid..

AMQ9197 Data length is larger than the exit buffer length.

Explanation: The data length '&1' set by exit '&3' is larger than the exit buffer length. The user exit returned data in the supplied exit buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

User action: Investigate why the user exit program set a data length that is not valid.

AMQ9201 Allocate failed to host '&3'.

Explanation: The attempt to allocate a conversation using &4 to host '&3' was not successful.

User action: The error may be due to an incorrect entry in the &4 parameters contained in the channel definition to host '&3'. Correct the error and try again. If the error persists, record the error values and contact your systems administrator. The return code from &4 was &1 (X'&2'). It may be possible that the listening program at host '&3' is not running. If this is the case, perform the relevant operations to start the listening program for protocol &4 and try again.

AMQ9202 Remote host '&3' not available, retry later.

Explanation: The attempt to allocate a conversation using &4 to host '&3' was not successful. However the error may be a transitory one and it may be possible to successfully allocate a &4 conversation later.

User action: Try the connection again later. If the failure persists, record the error values and contact your systems administrator. The return code from &4 is &1 (X'&2'). The reason for the failure may be that this host cannot reach the destination host. It may also be possible that the listening program at host '&3' was not running. If this is the case, perform the relevant operations to start the &4 listening program, and try again.

AMQ9203 A configuration error for &4 occurred.

Explanation: Allocation of a &4 conversation to host '&3' was not possible.

User action: The configuration error may be one of the following: 1. If the communications protocol is LU6.2, it may be that one of the transmission parameters (Mode, or TP Name) is incorrect. Correct the error and try again. The mode name should be the same as the mode defined on host &3. The TP name on &3 should be defined. 2. If the communications protocol is LU6.2, it may be that an LU6.2 session has not been established. Contact your systems administrator. 3. If the communications protocol is TCP/IP, it may be that the host name specified is incorrect. Correct the error and try again. 4. If the communications protocol is TCP/IP, it may be that the host name specified cannot be resolved to a network address. The host name may not be in the nameserver. The return code from &4 is &1 (X'&2'). Record the error values and tell the system administrator.

AMQ9204 Connection to host '&3' rejected.

Explanation: Connection to host '&3' over &4 was rejected.
User action: The remote system might not be configured to allow connections from this host. Check the &4 listener program has been started on host '&3'. If the conversation uses LU6.2, it is possible that either the userid or password supplied to the remote host is incorrect. If the conversation uses TCP/IP, it is possible that the remote host does not recognize the local host as a valid host. The return code from &4 is &1 X('&2'). Record the values and tell the systems administrator.

AMQ9205 The host name supplied is not valid.

Explanation: The supplied &4 host name '&3' could not be resolved into a network address. Either the name server does not contain the host, or the name server was not available.
User action: Check the &4 configuration on your host.

AMQ9206 Error on send to host '&3'.

Explanation: An error occurred sending data over &4 to '&3'. This may be due to a communications failure.
User action: Record the value &1 and the return code &4 and tell your systems administrator.

AMQ9207 The data received from host '&3' is not valid.

Explanation: Incorrect data format received from host '&3' over &4. It may be that an unknown host is attempting to send data. An FFST file has been generated containing the invalid data received.
User action: Tell the systems administrator.

AMQ9208 Error on receive from host '&3'.

Explanation: An error occurred receiving data from '&3' over &4. This may be due to a communications failure.
User action: Record the &4 return code &1 (X'&2') and tell the systems administrator.

AMQ9209 Connection to host '&3' closed.

Explanation: An error occurred receiving data from '&3' over &4. The connection to the remote host has unexpectedly terminated.
User action: Tell the systems administrator.

AMQ9210 Remote attach failed.

Explanation: There was an incoming attach from a remote host but the local host could not complete the bind.
User action: Record the &4 return code &1 (X'&2') and tell the systems administrator who should check the &4 configuration.

AMQ9211 Error allocating storage.

Explanation: The program was unable to obtain enough storage.
User action: Stop some programs which are using storage and retry the operation. If the problem persists contact your Systems Administrator.

AMQ9212 A TCP/IP socket could not be allocated.

Explanation: A TCP/IP socket could not be created, possibly because of a storage problem.
User action: Try the program again. If the failure persists record the value &1 and tell the systems administrator.

AMQ9213 A communications error for &4 occurred.

Explanation: An unexpected error occurred in communications.
User action: The return code from the &4&3 call was &1 (X'&2'). Record these values and tell the systems administrator.

AMQ9214 Attempt to use an unsupported communications protocol.

Explanation: An attempt was made to use an unsupported communications protocol type &2.
User action: Check the channel definition file. It may be that the communications protocol entered is not a currently supported one.

AMQ9215 Communications subsystem unavailable.

Explanation: An attempt was made to use the communications subsystem, but it has not been started.
User action: Start the communications subsystem, and rerun the program.

AMQ9216 Usage: &3 [-m QMgrName] [-n TPName]

Explanation: Values passed to the responder channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-n TPName] Default values will be used for parameters not supplied.
User action: Correct the parameters passed to the Channel program and retry the operation.

AMQ9217 The TCP/IP listener program could not be started.

Explanation: An attempt was made to start a new instance of the listener program, but the program was rejected.
User action: The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the listener program later).

AMQ9218 The TCP/IP listener program could not bind to port number &1.

Explanation: An attempt to bind the TCP/IP socket to the listener port was unsuccessful.
User action: The failure could be due to another program using the same port number. Record the return code &2 from the bind and tell the systems administrator.

AMQ9219 The TCP/IP listener program could not create a new connection for the incoming conversation.

Explanation: An attempt was made to create a new socket because an attach request was received, but an error occurred.
User action: The failure may be transitory, try again later. If the problem persists, record the return code &1 and tell the systems administrator. It may be necessary to free some jobs, or restart the communications system.

AMQ9220 The &4 communications program could not be loaded.

Explanation: The attempt to load the &4 library or procedure '&3' failed with error code &1.

User action: Either the library must be installed on the system or the environment changed to allow the program to locate it.

AMQ9221 Unrecognized protocol was specified.

Explanation: The specified value of '&3' was not recognized as one of the protocols supported.

User action: Correct the parameter and retry the operation.

AMQ9222 Cannot find the configuration file.

Explanation: The configuration file '&3' cannot be found. This file contains default definitions for communication parameters. Default values will be used.

User action: None.

AMQ9223 Enter a protocol type.

Explanation: The operation you are performing requires that you enter the type of protocol.

User action: Add the protocol parameter and retry the operation.

AMQ9224 Unexpected token detected.

Explanation: On line &1 of the INI file keyword '&3' was read when a keyword was expected.

User action: Correct the file and retry the operation.

AMQ9225 File syntax error.

Explanation: A syntax error was detected on line &1 while processing the INI file.

User action: Correct the problem and retry the operation.

AMQ9226 Usage: &3 [-m QMgrName] -t (TCP | LU62 | NETBIOS) [ProtocolOptions]

Explanation: Values passed to the listener program are invalid. The parameter string passed to this program is as follows :- [-m QMgrName] (-t TCP [-p Port] | -t LU62 [-n TPName] | -t NETBIOS [-l LocalName] [-e Names] [-s Sessions] [-o Commands] [-a Adaptor]) Default values will be used for parameters not supplied.

User action: Correct the parameters passed to the listener program and retry the operation.

AMQ9227 &3 local host name not provided.

Explanation:

User action: Add a local name to the configuration file and retry the operation.

AMQ9228 The &4 responder program could not be started.

Explanation: An attempt was made to start an instance of the responder program, but the program was rejected.

User action: The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the responder program later).

AMQ9229 The application has been ended.

Explanation: You have issued a request to end the application.

User action: None.

AMQ9230 An unexpected &4 event occurred.

Explanation: During the processing of network events, an unexpected event &1 occurred.

User action: None.

AMQ9231 The supplied parameter is not valid.

Explanation: The value of the &4 &5 parameter has the value '&3'. This value has either not been specified or has been specified incorrectly.

User action: Check value of the &5 parameter and correct it if necessary. If the fault persists, record the return code (&1,&2) and &4 and tell the systems administrator.

AMQ9232 No &3 specified

Explanation: The operation requires the specification of the &3 field.

User action: Specify the &3 and retry the operation.

AMQ9233 Error creating Listener thread for &3.

Explanation: The process attempted to create a new thread for an incoming connection.

User action: Contact the systems administrator.

AMQ9235 The supplied Local LU was invalid.

Explanation: The &4 Local LU name '&3' was invalid.

User action: Either the Local LU name was entered incorrectly or it was not in the &4 communications configuration. Correct the error and try again.

AMQ9236 The supplied Partner LU was invalid.

Explanation: The &4 Partner LU name '&3' was invalid.

User action: Either the Partner LU name was entered incorrectly or it was not in the &4 communications configuration. Correct the error and try again.

AMQ9238 A communications error for &4 occurred.

Explanation: An unexpected error occurred in communications.

User action: The return code from the &4&3 call was &1 with associated &5 &2.

Programmer response: Record the error values and tell the systems administrator.

AMQ9501 Usage: &3 [-m QMgrName] -c ChName.

Explanation: Values passed to the channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] -c ChName Default values will be used for parameters not supplied.

User action: Correct the parameters passed to the Channel program and retry the operation.

AMQ9502 Type of channel not suitable for action requested.

Explanation: The operation requested cannot be performed on channel '&3'. Some operations are only valid for certain channel types. For example, you can only ping a channel from the end sending the message.

User action: Check whether the channel name is specified correctly. If it is check that the channel has been defined correctly.

AMQ9503 Channel negotiation failed.

Explanation: Channel '&3' between this machine and the remote machine could not be established due to a negotiation failure.

User action: Tell the systems administrator who should look at the log on the remote system for messages explaining the cause of the negotiation failure.

AMQ9504 A protocol error was detected for channel '&3'.

Explanation: During communications with the remote queue manager, the channel program detected a protocol error. The failure type was &1 with associated data of &2.

User action: Contact the systems administrator who should examine the error logs to determine the cause of the failure.

AMQ9505 Channel sequence number wrap values are different.

Explanation: The sequence number for channel '&3' is &1, but the value specified at the remote location is &2. The two values must be the same before the channel can be started.

User action: Change either the local or remote channel definitions so that the values specified for the message sequence number wrap values are the same.

AMQ9506 Message receipt confirmation failed.

Explanation: Channel '&3' has ended because the remote queue manager did not accept the last batch of messages.

User action: The error log for the channel at the remote site will contain an explanation of the failure. Contact the remote Systems Administrator to resolve the problem.

AMQ9507 Channel '&3' is currently in-doubt.

Explanation: The requested operation cannot complete because the channel is in-doubt with host '&4'.

User action: Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or use the RESOLVE CHANNEL command to correct the problem manually.

AMQ9508 Program cannot connect to the queue manager.

Explanation: The connection attempt to queue manager '&4' failed with reason code &1.

User action: Ensure that the queue manager is available and operational.

AMQ9509 Program cannot open queue manager object.

Explanation: The attempt to open either the queue or queue manager object '&4' on queue manager '&5' failed with reason code &1.

User action: Ensure that the queue is available and retry the operation.

AMQ9510 Messages cannot be retrieved from a queue.

Explanation: The attempt to get messages from queue '&4' on queue manager '&5' failed with reason code &1.

User action: Ensure that the required queue is available and operational.

AMQ9511 Messages cannot be put to a queue.

Explanation: The attempt to put messages to queue '&4' on queue manager '&5' failed with reason code &1.

User action: Ensure that the required queue is available and operational.

AMQ9512 Ping operation is not valid for channel '&3'.

Explanation: Ping may only be issued for SENDER or SERVER channel types.

User action: If the local channel is a receiver channel, you must issue the ping from the remote queue manager.

AMQ9513 Maximum number of channels reached.

Explanation: The maximum number of channels that can be in use simultaneously has been reached.

User action: Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The number of permitted channels is a configurable parameter in the queue manager configuration file.

AMQ9514 Channel '&3' is in use.

Explanation: The requested operation failed because channel '&3' is currently active.

User action: Either end the channel manually, or wait for it to close, and retry the operation.

AMQ9515 Channel '&3' changed.

Explanation: The statistics shown are for the channel requested, but it is a new instance of the channel. The previous channel instance has ended.

User action: None.

AMQ9516 File error occurred.

Explanation: The filesystem returned error code &1 for file '&3'.

User action: Record the name of the file '&3' and tell the systems administrator, who should ensure that file '&3' is correct and available.

AMQ9517 File damaged.

Explanation: The program has detected damage to the contents of file '&3'.

User action: Record the values and tell the systems administrator who must restore a saved version of file '&3'. The return code was '&1' and the record length returned was '&2'.

AMQ9518 File '&3' not found.

Explanation: The program requires that the file '&3' is present and available.

User action: Record the name of the file and tell the systems administrator who must ensure that file '&3' is available to the program.

AMQ9519 Channel '&3' not found.

Explanation: The requested operation failed because the program could not find a definition of channel '&3'.

User action: Check that the name is specified correctly and the channel definition is available.

AMQ9520 Channel not defined remotely.

Explanation: There is no definition of channel '&3' at the remote location.

User action: Add an appropriate definition to the remote hosts list of defined channels and retry the operation.

AMQ9521 Host is not supported by this channel.

Explanation: The connection across channel '&5' was refused because the remote host '&4' did not match the host '&3' specified in the channel definition.

User action: Update the channel definition, or remove the explicit mention of the remote machine connection name.

AMQ9522 Error accessing the status table.

Explanation: The program could not access the channel status table.

AMQ9523 Remote host detected a protocol error.

Explanation: During communications through channel '&3', the remote queue manager channel program detected a protocol error. The failure type was &1 with associated data of &2.

User action: Tell the systems administrator, who should examine the error files to determine the cause of the failure.

AMQ9524 Remote queue manager unavailable.

Explanation: Channel '&3' cannot start because the remote queue manager is not currently available.

User action: Either start the remote queue manager, or retry the operation later.

AMQ9525 Remote queue manager is ending.

Explanation: Channel '&3' is closing because the remote queue manager is ending.

User action: None.

AMQ9526 Message sequence number error for channel '&3'.

Explanation: The local and remote queue managers do not agree on the next message sequence number. A message with sequence number &1 has been sent when sequence number &2 was expected.

User action: Determine the cause of the inconsistency. It could be that the synchronization information has become damaged, or has been backed out to a previous version. If the situation cannot be resolved, the sequence number can be manually reset at the sending end of the channel using the RESET CHANNEL command.

AMQ9527 Cannot send message through channel '&3'.

Explanation: The channel has closed because the remote queue manager cannot receive a message.

User action: Contact the systems administrator who should examine the error files of the remote queue manager, to determine why the message cannot be received, and then restart the channel.

AMQ9528 User requested closure of channel '&3'.

Explanation: The channel is closing because of a request by the user.

User action: None.

AMQ9529 Target queue unknown on remote host.

Explanation: Communication using channel '&3' has ended because the target queue for a message is unknown at the remote host.

User action: Ensure that the remote host contains a correctly defined target queue, and restart the channel.

AMQ9530 Program could not inquire queue attributes.

Explanation: The attempt to inquire the attributes of queue '&4' on queue manager '&5' failed with reason code &1.

User action: Ensure that the queue is available and retry the operation.

AMQ9531 Transmission queue specification error.

Explanation: Queue '&4' identified as a transmission queue in the channel definition '&3' is not a transmission queue.

User action: Ensure that the queue name is specified correctly. If so, alter the queue usage parameter of the queue to that of a transmission queue.

AMQ9532 Program cannot set queue attributes.

Explanation: The attempt to set the attributes of queue '&4' on queue manager '&5' failed with reason code &1.

User action: Ensure that the queue is available and retry the operation.

AMQ9533 Channel '&3' is not currently active.

Explanation: The channel was not stopped because it was not currently active.

User action: None.

AMQ9534 Channel '&3' is currently not enabled.

Explanation: The channel program ended because the channel is currently not enabled.

User action: Issue the START CHANNEL command to re-enable the channel.

AMQ9535 User exit not valid.

Explanation: Channel program '&3' ended because user exit '&4' is not valid.

User action: Ensure that the user exit is specified correctly in the channel definition, and that the user exit program is correct and available.

AMQ9536 Channel ended by an exit.

Explanation: Channel program '&3' was ended by exit '&4'.

User action: None.

AMQ9537 Usage: &3 [-m QMgrName] [-q InitQ]

Explanation: Values passed to the Channel initiator program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-q InitQ] Default values will be used for parameters not supplied.

User action: Correct the parameters passed to the program and retry the operation.

AMQ9538 Commit control error.

Explanation: An error occurred when attempting to start commitment control. Either exception '&3' was received when querying commitment status, or commitment control could not be started.

User action: Refer to the error log for other messages pertaining to this problem.

AMQ9539 No channels available.

Explanation: The channel initiator program received a trigger message to start an MCA program to process queue '&3'. The program could not find a defined, available channel to start.

User action: Ensure that there is a defined channel, which is enabled, to process the transmission queue.

AMQ9540 Commit failed.

Explanation: The program ended because return code &1 was received when an attempt was made to commit change to the resource managers. The commit ID was '&3'.

User action: Tell the systems administrator.

AMQ9541 CCSID supplied for data conversion not supported.

Explanation: The program ended because, either the source CCSID '&1' or the target CCSID '&2' is not valid, or is not currently supported.

User action: Correct the CCSID that is not valid, or ensure that the requested CCSID can be supported.

AMQ9542 Queue manager is ending.

Explanation: The program will end because the queue manager is quiescing.

User action: None.

AMQ9543 Status table damaged.

Explanation: The channel status table has been damaged.

User action: End all running channels and issue a DISPLAY CHSTATUS command to see the status of the channels. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9544 Messages written to the 'dead-letter queue'.

Explanation: During the processing of channel '&3' one or more messages have been put to a dead-letter queue. The location of the messages is &1, where 1 is the local dead-letter queue and 2 is the remote dead-letter queue.

User action: Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed. The program identifier (PID) of the processing program was '&4'.

AMQ9545 Disconnect interval expired.

Explanation: Channel '&3' closed because no messages arrived on the transmission queue within the disconnect interval period.

User action: None.

AMQ9546 Error return code received.

Explanation: The program has ended because return code &1 was returned from an internal function.

User action: Correct the reason for the failure and retry the operation.

AMQ9547 Type of remote channel not suitable for action requested.

Explanation: The operation requested cannot be performed because channel '&3' on the remote machine is not of a suitable type. For example, if the local channel is defined as a sender the remote machine must define its channel as either a receiver or requester.

User action: Check that the channel name is specified correctly. If it is, check that the remote channel has been defined correctly.

AMQ9548 Message put to the 'dead-letter queue'.

Explanation: During processing a message has been put to the dead-letter queue.

User action: Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

AMQ9549 Transmission Queue '&3' inhibited for MQGET.

Explanation: An MQGET failed because the transmission queue had been previously inhibited for MQGET.

User action: None.

AMQ9550 Channel program &3 cannot be stopped at this time.

Explanation: The channel program is currently busy and cannot be stopped at the moment.

User action: Issue the STOP CHANNEL command again at a later time.

AMQ9551 Protocol not supported by remote host

Explanation: The operation you are performing over Channel '&3' to the host at '&4' is not supported by the target host.

User action: Check that the connection name parameter is specified correctly and that the levels of the products in use are compatible.

AMQ9552 Security flow not received.

Explanation: During communications through channel '&3' the local security exit requested security data from the remote machine. The security data has not been received so the channel has been closed.

User action: Tell the systems administrator who should ensure that the security exit on the remote machine is defined correctly.

AMQ9553 Not supported.

Explanation: The command or function attempted is not currently supported on this platform.

User action: None.

AMQ9554 User not authorized.

Explanation: You are not authorized to perform the Channel operation.

User action: Tell the systems administrator who should ensure that the correct access permissions are available to you, and then retry the operation.

AMQ9555 File format error.

Explanation: The file '&3' does not have the expected format.

User action: Ensure that the file name is specified correctly.

AMQ9556 Channel synchronization file missing or damaged.

Explanation: The channel synchronization file '&3' is missing or does not correspond to the stored channel information for queue manager '&4'.

User action: Rebuild the synchronization file using the rcrmqobj command `rcrmqobj -t syncfile (-m q-mgr-name)`

AMQ9557 Queue Manager UserID initialization failed.

Explanation: The call to initialize the user ID failed with CompCode &1 and Reason &2.

User action: Correct the error and try again.

AMQ9558 Remote Channel is not currently available.

Explanation: The channel program ended because the channel '&3' is not currently available on the remote system. This could be because the channel is disabled or that the remote system does not have sufficient resources to run a further channel.

User action: Check the remote system to ensure that the channel is available to run and retry the operation.

AMQ9560 Rebuild Synchronization File - program started

Explanation: Rebuilding the Synchronization file for Queue Manager '&3'.

User action: None.

AMQ9561 Rebuild Synchronization File - program completed normally

Explanation: Rebuild Synchronization File program completed normally.

User action: None.

AMQ9562 Synchronization file in use.

Explanation: The Synchronization file '&3' is in use and cannot be recreated.

User action: Stop any channel activity and retry the rcrmqobj command.

AMQ9563 Synchronization file cannot be deleted

Explanation: The filesystem returned error code &1 for file '&3'.

User action: Tell the systems administrator who should ensure that file '&3' is available and not in use.

AMQ9564 Synchronization File cannot be created

Explanation: The filesystem returned error code &1 for file '&3'.

User action: Tell the systems administrator.

AMQ9565 No dead-letter queue defined.

Explanation: The queue manager '&4' does not have a defined dead-letter queue.

User action: Either correct the problem that caused the program to try and write a message to the dead-letter queue or create a dead-letter queue for the queue manager.

AMQ9566 Invalid MQSERVER value

Explanation: The value of the MQSERVER environment variable was '&3'. The variable should be in the format 'ChannelName/Protocol/ConnectionName'.

User action: Correct the MQSERVER value and retry the operation.

AMQ9572 Message header is not valid.

Explanation: Channel '&3' is stopping because a message header is not valid. During the processing of the channel, a message was found that has a header that is not valid. The dead-letter queue has been defined as a transmission queue, so a loop would be created if the message had been put there.

User action: Correct the problem that caused the message to have a header that is not valid.

AMQ9573 Maximum number of active channels reached.

Explanation: There are too many channels active to start another. The current defined maximum number of active channels is &1.

User action: Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The maximum number of active channels is a configurable parameter in the queue manager configuration file.

AMQ9574 Channel &3 can now be started.

Explanation: Channel &3 has been waiting to start, but there were no channels available because the maximum number of active channels was running. One, or more, of the active channels has now closed so this channel can start.

User action:

AMQ9999 Channel program ended abnormally.

Explanation: Channel program '&3' ended abnormally.

User action: Look at previous error messages for channel program '&3' in the error files to determine the cause of the failure.

Appendix O. Notices

The following paragraph does not apply to any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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 of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact Laboratory Counsel, Mail Point 151, IBM United Kingdom Laboratories, Hursley Park, Winchester, Hampshire SO21 2JN, England. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Trademarks

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

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

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

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

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, and the Windows 95 Logo are trademarks of Microsoft Corporation.

Digital, OpenVMS, VAX, and AXP are trademarks of the Digital Equipment Corporation

Other company, product, and service names, which may be denoted by a double asterisk (**), may be 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

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

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

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

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

APAR. Authorized program analysis report.

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 open an MQSeries object.

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

authorization service. In MQSeries on UNIX systems, MQSeries for OS/2, and MQSeries for Windows NT, 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*.

basic mapping support (BMS). An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS. Basic mapping support.

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

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

C

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

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

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

channel definition file (CDF) • dead-letter queue handler

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.

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

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

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

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

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

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

command 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, and MQSeries for Windows NT, a file that contains configuration information related to, for example, logs, communications, or installable services. Synonymous with *.ini file*. See also *stanza*.

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

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

context. Information about the origin of a message.

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

control command. In MQSeries on UNIX systems, MQSeries for OS/2, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. 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 conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

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

DCE. Distributed Computing Environment.

DCI. Data conversion interface.

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

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and

processes messages on the queue in accordance with a user-written rules table.

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

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

Distributed Computing Environment (DCE).

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

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

DLQ. Dead-letter queue.

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

E

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

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

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

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

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

F

FFST. First Failure Support Technology.

FIFO. First-in-first-out.

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

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

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

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

G

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

H

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

I

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

.ini file. See *configuration file*.

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

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

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

installable services • message format service (MFS)

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

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

L

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

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

local definition. An MQSeries object 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.

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

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

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

M

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2, and MQSeries for Windows NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. (1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. (2) In system programming, information intended for the terminal operator or system administrator.

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

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

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

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

message format service (MFS). In IMS, and editing facility that allows application programs to deal with

simple logical messages, instead of device-dependent data, thus simplifying the application development process. See *message input descriptor* and *message output descriptor*.

message input descriptor (MID). In IMS, the MFS control block that describes the format of the data presented to the application program. Contrast with *message output descriptor*.

message output descriptor (MID). In IMS, the MFS control block that describes the format of the output data produced by the application program. Contrast with *message input descriptor*.

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

message queue. Synonym for *queue*.

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

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

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

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

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

MQI. Message queue interface.

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

MQSC. MQSeries commands.

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

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full

queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

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

N

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

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

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

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

NSI. Name service interface.

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

O

OAM. Object authority manager.

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

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

object descriptor. A data structure that identifies a particular MQSeries object. 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 • quiescing

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 is functioning.

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

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

principal. In MQSeries on UNIX systems, MQSeries for OS/2, and MQSeries for Windows NT, a term used for a user identifier. 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.

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.

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.

request message. A type of message used to request a reply from another program.

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

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

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

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

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

rollback. Synonym for *back out*.

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

S

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

SEI. Security enabling interface.

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

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

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

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

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

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

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

service interval event • two-phase commit

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*.

SIT. System initialization table.

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

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

system initialization table (SIT). A table containing parameters used by CICS on start up.

T

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

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

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

tranid. See *transaction identifier*.

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

transmission program. See *message channel agent*.

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

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

triggering. In MQSeries, a facility 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

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 identifier service (UIS). In MQSeries for OS/2 and MQSeries for Windows NT, 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.

Index

A

ACTION keyword, rules table 112
action keywords, rules table 112
administration
 authorizations 96
 command sets 33
 control commands 33
 MQSeries commands (MQSC) 34
 programmable command format commands (PCF) 35
 local 49
 remote 72
 channels 73
 objects 71
 transmission queues 73
 security, user groups for 31
alias queues 65
 authorizations to 90
 description 7
aliases
 queue manager 81
 reply-to queues 81
alter queue manager attributes 53
alternate user authority 90
amqscoma.tst
 creating default and system objects 44
 location 233
amqsdq, the sample DLQ handler 108
amqsvfcx sample program
 using with MQSeries for Digital OpenVMS 284
ancillary information 307
application
 data 4
 design considerations 156
 MQI administration support 49
 programming errors, examples of 152
 time-independent 3
APPLIDAT keyword, rules table 110
APPLNAME keyword, rules table 111
APPLTYPE keyword, rules table 111
Attachmate PathWay 299
attributes
 ALL attribute 59
 altering 53
 changing 61
 default 59
 displaying queue manager 51
 MQSC and PCFs compared 35
 queue manager
 altering 53
 displaying 51

attributes (*continued*)
 queues 7
authority
 alternate user 90
 commands 88
 context 91
 installable services 88
 set/reset command 211
authorization
 administration 96
 dspmqaut command 89
 lists 87
 MQI 93
 rights identifiers 86
 setmqaut command 89
authorization files
 all class 101
 authorization to 101
 class 100
 contents 99
 directories 99
 managing 101
 paths 98
 understanding 98

B

back out in-doubt messages
 OS/2 295
 UNIX systems 295
 Windows NT 295
batch-job log files 161
bibliography xii
browsing queues 62
building an exit with MQSeries for Digital OpenVMS 284
building your application
 on Digital OpenVMS 279

C

C language
 include files for MQSeries for Digital OpenVMS 279
case-sensitive control commands 33
CCSID 28, 235
CCSID language support tables 241
ccsid.tbl 28
cell, DCE and queues 103
changing queue attributes 61
channel
 change definition 292
 channel control function
 Digital OpenVMS systems 285

- channel (*continued*)
 - channel control function (*continued*)
 - OS/2 285
 - UNIX systems 285
 - Windows NT 285
 - command security requirements 92
 - commands 92
 - configuration 141
 - create definition 292
 - creating 288
 - defining 74
 - delete channel 292
 - description 10, 71
 - display
 - Digital OpenVMS systems 292
 - OS/2 292
 - UNIX systems 292
 - Windows NT 292
 - display status
 - Digital OpenVMS systems 292
 - OS/2 292
 - UNIX systems 292
 - Windows NT 292
 - displaying 289
 - displaying settings 292
 - displaying status 292
 - escape command authorizations 96
 - events 121
 - OS/2
 - resolve 295
 - ping
 - Digital OpenVMS systems 293
 - OS/2 293
 - UNIX systems 293
 - Windows NT 293
 - remote administration 73
 - renaming 290
 - Reset
 - OS/2 295
 - UNIX systems 295
 - Windows NT 295
 - resolve 295
 - run command 202
 - run initiator command 201
 - security 92
 - start 290
 - OS/2 293
 - UNIX systems 293
 - Windows NT 293
 - starting 75
 - stopping
 - OS/2 294
 - UNIX systems 294
 - Windows NT 294
 - UNIX systems
 - resolve 295
- channel (*continued*)
 - Windows NT
 - resolve 295
 - channel control function
 - Digital OpenVMS systems 285
 - OS/2 285
 - UNIX systems 285
 - Windows NT 285
 - channel definition file
 - Digital OpenVMS 285
 - OS/2 285
 - UNIX systems 285
 - Windows NT 285
 - channel functions
 - OS/2 292
 - UNIX systems 292
 - Windows NT 292
 - character set, specifying 28
 - Chinese language support 265, 266
 - CICS
 - user ID 89
 - circular logging 126
 - Cisco MultiNet for OpenVMS 298
 - clearing a local queue 61
 - clients 11
 - error messages on DOS and Windows 167
 - installation 26
 - problem determination 166
 - trigger monitor start command 209
 - COBOL
 - copy files for MQSeries for Digital OpenVMS 281
 - using named constants with MQSeries for Digital OpenVMS 282
 - code-page conversions 241
 - coded character set
 - identifier 235
 - on OpenVMS 28
 - specifying 28
 - table 29
 - codeset 235
 - command files 54
 - command queue 9
 - command server
 - display command 186
 - displaying status 47
 - end command 190
 - remote administration 47
 - start command 217
 - starting 47
 - stopping 48
 - command set
 - administration 33
 - comparison 36
 - commands
 - case sensitivity 23
 - comparison of sets 36

commands (*continued*)
 control 33
 create queue manager (crtmqm) 176
 data conversion (crtmqcvx) 174
 delete queue manager (dlmqm) 180
 display authority (dspmqaut) 182
 display command server (dspmqcsv) 186
 display MQSeries files (dspmqfls) 187
 display MQSeries formatted trace (dspmqtrc) 189
 end command server (endmqcsv) 190
 end MQSeries trace (endmqtrc) 194
 end queue manager (endmqm) 192
 help with syntax 172
 MQSC
 ALTER QLOCAL 61
 ALTER QREMOTE 80
 DEFINE CHANNEL 74
 DEFINE QALIAS 65
 DEFINE QLOCAL 60
 DEFINE QLOCAL LIKE 60
 DEFINE QLOCAL REPLACE 61
 DEFINE QMODEL 67
 DEFINE QREMOTE 78
 DELETE QLOCAL 61
 DELETE QREMOTE 80
 DISPLAY QREMOTE 80
 MQSC command files 54
 input 54
 output reports 54
 MQSeries (MQSC)
 using 34
 verifying 56
 MQSeries commands (MQSC) 34
 programmable command format (PCF) 35
 record media image (rcdmqimg) 195
 recreate object (rcrmqobj) 197
 resolve MQSeries transactions (rsvmqtrn) 199
 run channel (runmqchl) 202
 run channel initiator (runmqchi) 201
 run dead-letter queue handler 203
 run DLQ handler (runmqdlq) 107
 run listener (runmqlsr) 205
 run MQSeries commands (runmqsc) 206
 runmqsc 51
 security commands
 dspmqaut 89
 setmqaut 87
 set/reset authority (setmqaut) 87, 211
 start client trigger monitor (runmqtmc) 209
 start command server (strmqcsv) 217
 start MQSeries trace (strmqtrc) 219
 start queue manager (strmqm) 218
 start trigger monitor (runmqtrm) 210
 commands for MQSeries 33
 commit in-doubt messages
 OS/2 295
 commit in-doubt messages (*continued*)
 UNIX systems 295
 Windows NT 295
 committed messages
 OS/2 295
 UNIX systems 295
 Windows NT 295
 compiling
 for MQSeries for Digital OpenVMS 283
 configuration
 size and location of log 143
 system 21
 configuration files
 LogPrimaryFile value 145
 MQSeries (mqs.ini)
 contents 139
 LogBufferPages 145
 LogDefaultPath 146
 LogDefaults 144
 LogFilePages 145
 logging parameters 146
 LogPath 146
 LogSecondaryFiles 145
 overview 139
 path 56
 overview 139
 queue manager (qm.ini)
 contents 141
 disabling the object authority manager 86
 Log stanza 144
 LogBufferPages 145
 LogDefaultPath 146
 LogFilePages 145
 logging parameters 146
 LogPath 146
 LogSecondaryFiles 145
 stanzas 141
 connection
 DECnet Phase IV 305
 DECnet Phase V 305
 defining LU 6.2
 Digital OpenVMS systems 300
 LU 6.2
 Digital OpenVMS systems 297
 TCP/IP
 Digital OpenVMS systems 297
 constants in COBOL for MQSeries for Digital
 OpenVMS 282
 contents of
 mqs.ini 139
 qm.ini 141
 contents of COBOL copy files
 with MQSeries for Digital OpenVMS 280
 context authority 91
 control commands 33
 case-sensitive 33

- control commands (*continued*)
 - runmqsc 51
- controlled shutdown 45
- conversions, code-page 241
- copy files
 - for MQSeries for Digital OpenVMS 280
- Correlld, performance considerations when using 157
- creating
 - a queue manager 22
 - crtmqm command 176
 - default objects 44
 - process definitions 70
 - queue manager 39, 44
 - system objects 44
- Creating a channel 288
 - Digital OpenVMS 288
 - OS/2 288
 - UNIX systems 288
 - Windows NT 288
- creating objects 288
 - Digital OpenVMS 288
 - OS/2 288
 - UNIX systems 288
 - Windows NT 288
- crtmqcvx command
 - examples 174
 - parameters 174
 - return codes 174
- crtmqm command 176
 - examples 179
 - parameters 176
 - related commands 179
 - return codes 178
- current queue depth 60
- customizing
 - options available 28
 - overview
 - data conversion 28
 - defining objects 29

D

- Danish language support 245
- data conversion
 - crtmqcvx command 174
 - customizing 28
- data types
 - for MQSeries for Digital OpenVMS 280
- DCE
 - cell 103
 - configuration 104
 - sharing queues 103
- dead-letter header, MQDLH 107
- dead-letter queue
 - description 8
 - handler 203
- dead-letter queue (*continued*)
 - specifying 41
- debugging
 - common programming errors 152
 - preliminary checks 149
 - secondary checks 153—156
- DECnet Phase IV 297
- DECnet Phase IV connection 305
- DECnet phase V connection 306
- default
 - attributes of objects 59
 - objects 10
 - creating 44
 - defining 29
 - overriding the configuration file 143
 - prefix 30
 - queue manager 40
 - accidental change 46
 - accidental deletion 176
 - changing 46, 52
 - commands processed 51
 - rights identifier for authority 85
 - system objects 227
 - transmission queue 41
- default transmission queue 80
- DEFINE QUEUE command, REPLACE attribute
- defining
 - an LU 6.2 connection
 - Digital OpenVMS systems 300
- defining queues 7
- deleting 61
 - dltmqm command 180
 - local queue 61
 - queue manager 46
- desktop clients 13
- DESTQ keyword, rules table 111
- DESTQM keyword, rules table 111
- Digital TCP/IP services for OpenVMS 298
- directories 90
 - authorization 99
 - queue manager 90
- directory structure 229
- disabled receiver channels 293
- disabling events 122
- disabling the object authority manager 86
- disk quotas 15
- disk requirements for installation 14
- display 292
 - authority command 182
 - command server command 186
 - MQSeries files command 187
 - MQSeries formatted trace output command 189
 - process definitions 70
 - queue manager attributes 51
 - status of command server 47

- display authority command
 - See dspmqaut command
- display channel
 - OS/2 289
 - UNIX systems 289
 - Windows NT 289
- Display channel status 290
 - Digital OpenVMS 290
 - OS/2 290
 - UNIX systems 290
 - Windows NT 290
- distributed queuing
 - dead-letter queue 9
 - incorrect output 160
 - undelivered-message queue 9
- DLQ handler
 - invoking 107
 - rules table 108
 - sample, amqsdq 108
- dltmqm command 180
 - examples 180
 - parameters 180
 - related commands 181
 - return codes 180
- DOS clients error messages 167
- dspmqaut command 182
 - examples 185
 - parameters 182
 - related commands 185
 - return codes 184
 - using 87, 88
- dspmqcsv command 186
 - examples 186
 - parameters 186
 - related commands 186
 - return codes 186
- dspmqfls command 187
 - examples 188
 - parameters 187
 - return codes 188
- dspmqtrc command 189
 - parameters 189
 - related commands 189
- dynamic queues 5
 - authorizations to 90

E

- edit
 - change
 - Digital OpenVMS systems 292
 - OS/2 292
 - UNIX systems 292
 - Windows NT 292
 - create
 - OS/2 292
 - UNIX systems 292

- edit (*continued*)
 - create (*continued*)
 - Windows NT 292
 - delete
 - Digital OpenVMS systems 292
 - OS/2 292
 - UNIX systems 292
 - Windows NT 292
 - enabling
 - events 122
 - security 86
 - end MQSeries trace 194
 - ending a channel 294
 - ending a queue manager 45
 - ending interactive MQSC commands 51
 - ending SNA Listener process 303
 - endmqcsv command 190
 - examples 190
 - parameters 190
 - related commands 191
 - return codes 190
 - endmqm command 45, 192
 - examples 193
 - parameters 192
 - related commands 193
 - return codes 193
 - endmqtrc command 194
 - examples 194
 - parameters 194
 - related commands 194
 - return codes 194
 - error log 160
 - error occurring before established 162
 - example 163
 - error logs 294
 - error messages 51
 - escape PCFs 35
 - event queue 9
 - event-driven processing 3
 - events
 - channel 121
 - instrumentation
 - description 119
 - enabling and disabling 122
 - message 123
 - types of 121
 - what they are 119
 - why use them 120
 - queues 122
 - trigger 122
 - types of 121
 - examples
 - crtmqcvx command 174
 - crtmqm command 179
 - dltmqm command 180
 - dspmqaut command 185

examples (*continued*)

- dspmqlsv command 186
- dspmqlfs command 188
- endmqcsv command 190
- endmqm command 193
- endmqtrc command 194
- error log 163
- programming errors 152
- rcdmqimg command 196
- rcrmqobj command 198
- runmqsc command 207
- setmqaut command 216
- strmqcsv command 217
- strmqm command 218
- strmqtrc command 220

exits

- user with MQSeries for Digital OpenVMS 284

F

- fast messages 237
- feedback from MQSC commands 51
- FEEDBACK keyword, rules table 111
- FFST, examining 165
- file paths to objects 30
- file sizes, for logs 147
- files
 - authorization
 - all class 101
 - authorizations to 101
 - class 100
 - contents 99
 - managing 101
 - paths 98
 - understanding 98
 - configuration
 - in problem determination 164
 - overview 139
 - log control 126
 - MQSeries configuration 139
 - queue manager configuration 141
 - understanding names 42
- Finnish language support 246
- FORMAT keyword, rules table 111
- format of logs 126
- French language support 250
- functions available
 - Digital OpenVMS 286
 - OS/2 286
 - UNIX systems 286
 - Windows NT 286
- FWDQ keyword, rules table 112
- FWDQM keyword, rules table 112

G

- Gaelic language support 249
- German language support 244
- glossary 349

H

- hard disk requirements 14
- HEADER keyword, rules table 113
- help for syntax 172

I

- Icelandic language support 253
- in-doubt messages, commit or backout
 - OS/2 295
 - UNIX systems 295
 - Windows NT 295
- include files
 - for MQSeries for Digital OpenVMS 279
 - macro variables for MQSeries for Digital OpenVMS 280
- incorrect output 157
- initiation queue
 - defining 69
 - description 8
- input, redirecting 51
- INPUTQ keyword, rules table 109
- INPUTQM keyword, rules table 109
- installable component
 - authority manager (OAM) 84
 - name service 103
- installable services
 - disabling object authority manager 86
 - disabling 86
 - name service 103
 - object authority manager 84
- installation
 - clients 26
 - components 13
 - directory structure 21
 - procedure for Digital OpenVMS V6.2 15
 - requirements 14
 - supported code sets 28
 - unsuccessful 25
 - verifying 22
- instrumentation event
 - description 119
 - enabling 122
 - messages 123
 - types of 121
 - why use them 120
- interactive MQSC
 - ending 51
 - feedback from 51
 - using 51

issuing MQSeries commands 50
Italian language support 247

J

Japanese language support 263

K

Kanji language support 263
Katakana language support 263
Korean language support 264

L

libraries to use
 with MQSeries for Digital OpenVMS 283
LIKE attribute 60
linear logging 127
linking
 for MQSeries for Digital OpenVMS 283
listening on TCP/IP
 Digital OpenVMS systems 298
local administration 49
local queues
 clearing 61
 command 9
 copying definitions 60
 dead-letter 8
 defining one 58
 deleting 61
 description 7
 initiation 8
 transmission 8
 undelivered-message 8
locale 235
locations of objects 30
log
 configuration 143
 error 160
 error, example of 163
 file
 control 126
 path 146
 reuse 128
 size 145
 file size 147
 format 126
 managing 130
 number of buffers 145
 overheads 147
 parameters 30, 42
 primary files 145
 queue manager 125
 secondary files 145
 type of 145

log (*continued*)
 using for recovery 132
logging
 checkpoints 128
 circular 126
 linear 127
 media recovery 133
 types of 126
logical name, disabling security 86
logs for errors 294
LU 6.2 connection
 MQSeries for Digital OpenVMS systems 297

M

macro variables
 for MQSeries for Digital OpenVMS 280
managing access 85
managing log files 131
managing objects for triggering 68
maximum line length for MQSC commands 54
media images
 description 132
 record 133
 record command 195
 recovering 133
memory requirements 14
memory requirements for installation 14
message
 administration 326—336
 common service 315—317
 containing unexpected information 159
 description 4
 descriptor 4
 errors on DOS and Windows clients 167
 for instrumentation events 123
 format 309
 groups 310
 information 309
 installable services 311—314
 lengths of 4
 MQSeries product 318—325
 not appearing on queues 158
 operator 162
 performance considerations
 lengths of 156
 persistent 156
 queuing 3
 remote 337—343
 retrieval algorithms 5
 searching for particular 157
 structure 309
 translated 22
 undelivered 163
 variable length 157
 variables 309

- message length, decreasing 61
- message queue interface (MQI) 3
- message queuing 3
- message-driven processing 3
- Messages
 - backout in-doubt messages 295
 - commit in-doubt messages 295
 - resolve in-doubt messages 295
- model queues
 - defining 67
 - description 8
 - working with 67
- monitoring
 - queue managers 120
- Monitoring and controlling DQM
 - Digital OpenVMS systems 285
 - OS/2 285
 - UNIX systems 285
 - Windows NT 285
- MQ*_DEFAULT values
 - with MQSeries for Digital OpenVMS 280
- MQDATA 167
- MQDLH, dead-letter header 107
- MQI
 - authorizations 93
 - description 3
 - local administration support 49
 - queue manager calls 7
- mqm
 - rights identifier 83
 - user ID 89
- MQOPEN authorizations 93
- MQPUT and MQPUT1, performance
 - considerations 157
- MQPUT authorizations 93
- mqs.ini
 - See configuration files
- mqs.ini, path to 56
- MQSC 51
 - attributes 35
 - command files
 - input 54
 - output reports 54
 - running 55
 - commands 34
 - ending interactive input 51
 - escape PCFs 35
 - how to issue commands 50
 - issuing commands interactively 51
 - issuing remotely 76
 - maximum line length 54
 - problems
 - local 56
 - remote 77
 - redirecting input and output 53
 - sample files 233
- MQSC (*continued*)
 - security requirements on channels 92
 - timed out command responses 76
 - using commands 53
 - verifying commands 56
- MQSC commands 51
 - ALTER QLOCAL 61
 - ALTER QREMOTE 80
 - DEFINE CHANNEL 74
 - DEFINE QALIAS 65
 - DEFINE QLOCAL 60
 - DEFINE QLOCAL LIKE 60
 - DEFINE QLOCAL REPLACE 61
 - DEFINE QMODEL 67
 - DEFINE QREMOTE 78
 - DELETE QLOCAL 61
 - DELETE QREMOTE 80
 - DISPLAY QREMOTE 80
 - issuing interactively 51
 - maximum line length 54
 - using 34
- MQSeries
 - overview for OpenVMS 225
 - rights identifier, MQM 83
- MQSeries configuration file
 - See configuration files
- MQSeries for Digital OpenVMS
 - <cmqc.h> 279
 - <cmqfc.h> 279
 - <cmqxc.h> 279
 - <cmqzc.h> 279
 - building user exit 284
 - building your application 279
 - C compiler 283
 - CMQDLHL 281
 - CMQDLHV 281
 - CMQGMOL 281
 - CMQGMOV 281
 - CMQMDL 281
 - CMQMDV 281
 - CMQODL 281
 - CMQODV 281
 - CMQPMOL 281
 - CMQPMOV 281
 - CMQTM2L 281
 - CMQTM2V 281
 - CMQTML 281
 - CMQTMV 281
 - CMQV 281
 - CMQXQHL 281
 - CMQXQHV 281
 - flags when compiling 283
 - header files 279
 - include files 279
 - link libraries 283

- MQSeries for Digital OpenVMS systems
 - problem solving 304
 - setting up communication 297
 - SNA onfiguration 300
- MQSeries publications xii
- MQSNOAUT logical name 86
- MQZAO constants and authority 94
- Msgld, performance considerations when using 157
- MSGTYPE keyword, rules table 111
- Multilingual language support 251
- MVS/ESA queue manager 76

N

- name service 103
- names
 - allowed for objects 171
 - objects 5
- naming conventions, national language support 171
- national language support
 - naming conventions 171
 - NLSPATH environment variable 22
- nobody, default rights identifier 85
- Norwegian language support 245
- notification of events 122

O

- OAM 84
- object authority manager 84
 - default rights identifier 85
 - disabling 86
 - dspmqaout command 89
 - how it works 85
 - principals 85
 - sensitive operations 89
 - setmqaut command 87
- objects
 - access to 83
 - creating 288
 - customizing 29
 - default
 - attributes 59
 - creating 44
 - for triggering 68
 - media image 132
 - names 5
 - naming conventions 171
 - process definition 9
 - queue 7
 - queue manager
 - location 30
 - MQI calls 7
 - prefixes 30
 - recovery 133
 - recreate command 197

- objects (*continued*)
 - remote administration 71
 - system
 - creating 44
 - system default 10, 227
 - types 5
- OpenVMS
 - hardware required 225
 - overview of 225
 - software required 225
- OpenVMS logged-in user ID 89
- OpenVMS rights identifier
 - default, nobody 85
 - MQM 83
- operating system logical name, disabling security 86
- operator commands, no response from 153
- operator messages 162
- options
 - resolve 295
- output, redirecting 51
- overheads, for logs 147
- overrides
 - in configuration files 143
- overview of MQSeries for Digital OpenVMS 225

P

- parameters
 - runmqslr command 205
- pattern-matching keywords, rules table 110
- PCF
 - See programmable command format (PCF)
- performance considerations when using trace 164
- performance events 121
- permanent queues 5
- PERSIST keyword, rules table 111
- ping
 - OS/2 293
 - UNIX systems 293
 - Windows NT 293
- ping with LU 6.2 293
- Portuguese language support 252
- post-installation
- predefined queues 5
- preemptive queue manager shutdown 45
- prefix, default 30
- preparation
 - getting started
 - Digital OpenVMS 288
 - OS/2 288
 - UNIX systems 288
 - Windows NT 288
- primary group authorizations
- primary log files 145
- primary rights identifier, for authority 85

- principals
 - holding more than one rights identifier 85
 - managing access to 85
- problem determination 149
 - clients 166
 - configuration files 164
 - further checks 153—156
 - incorrect output
 - messages containing unexpected information 159
 - messages not appearing on queues 158
 - with distributed queuing 160
 - no response from commands 153
 - programming errors 152
 - things to check first 149—152
 - trace 164
- problems
 - recovering from 132
 - running MQSC commands 56
 - using MQSC locally 56
 - using MQSC remotely 77
- process definitions
 - creating 70
 - description 9
 - displaying 70
- process rights authorizations
- processing, event-driven 3
- programmable command format
 - See programmable command format (PCF)
- programmable command format (PCF)
 - administration with 35
 - attributes 35
 - authorizations 93
 - commands 35
 - description 35
 - escape PCFs 35
 - security requirements 92
- programming errors, examples of 152
- programs, samples supplied 233
- protected resources 86
- publications
 - MQSeries xii
- PUTAUT keyword, rules table 113

Q

- qm.ini
 - See configuration files
- queue depth
 - current 60
 - determining 60
- queue manager
 - alias
 - remote queue 81
 - authorization directories 99
 - authorizations 90
- queue manager (*continued*)
 - circular logging
 - restart recovery 127
 - command server 47
 - configuration file 141
 - configuration files
 - LogPath 146
 - specifying 42
 - configuration overview 31
 - creating 22, 39, 44
 - crtmqm command 176
 - default 40
 - accidental change 46
 - accidental deletion 176
 - changing 46
 - deleting 46
 - dltmqm command 180
 - description 6
 - directories 90
 - endmqm command 192
 - events 121
 - immediate shutdown 45
 - linear logging 127
 - local administration 49
 - logs 125
 - monitoring 120
 - name transformation 43
 - numbers of 40
 - object authority manager
 - description 84
 - disabling 86
 - objects
 - location 30
 - MQI calls 7
 - prefixes 30
 - on MVS/ESA 76
 - preemptive shutdown 45
 - recording media images 133
 - remote administration 71
 - removing
 - manually 276
 - restart 46
 - shutdown 45
 - controlled 45
 - quiesce 45
 - specifying on runmqsc 52
 - starting 44
 - stopping 45
 - manually 275
 - unique name 40
- queue manager configuration file
 - See configuration files
- queued mode, of runmqsc 76
- queues
 - alias 7
 - aliases, working with 65

queues (*continued*)
 application
 defining for triggering 68
 attributes 7
 changing 61
 authorizations to 90
 browsing 62
 command 9
 dead-letter 8
 specifying 41
 defining 7
 description 4
 distributed, incorrect output from 160
 dynamic 5
 event 9
 event notification 122
 for MQSeries applications 49
 initiation
 defining 69
 trigger messages 8
 local 7
 clearing 61
 copying 60
 defining 58
 deleting 61
 model 8
 defining 67
 working with 67
 objects
 alias 7
 local 7
 model 8
 remote 7
 predefined 5
 remote 7
 creating 78
 queue manager alias 81
 working with 81
 reply-to 9, 81
 shared
 configuration tasks 103
 shared on different queue managers 103
 temporary 5
 transmission 8
 creating 80
 default 41, 80
 defining 74
 remote administration 73
 undelivered-message 8
 specifying 41
 working with 58
 quiesce shutdown 45
 queue manager 45

R

railroad diagrams, how to read 171
 rcdmqimg command 195
 examples 196
 parameters 195
 related commands 196
 return codes 196
 rcrmobj command 197, 198
 examples 198
 parameters 197
 related commands 198
 return codes 198
 REASON keyword, rules table 111
 receiving on DECnet Phase IV 305
 receiving on TCP/IP
 Digital OpenVMS systems 298
 recovering
 media images 133
 recovery
 scenarios 136
 damaged queue manager object 137
 damaged single object 137
 disk drive failures 136
 redirecting input and output, on MQSC commands 53
 remote
 administration 72
 of objects 71
 issuing of MQSC commands 76
 queue definition
 creating 78
 queue object
 working with 81
 queues
 as queue manager aliases 81
 as reply-to queue aliases 81
 queuing
 description 71
 recommendations 77
 security considerations 91
 remote administration
 command server 47
 initial problems 77
 remote queues
 authorizations to 90
 description 7
 removing
 queue manager
 manually 276
 renaming a channel 290
 REPLACE attribute, DEFINE commands 54
 reply-to queue 9
 reply-to queue aliases 81
 REPLYQ keyword, rules table 111
 REPLYQM keyword, rules table 111

- requirements
 - disk storage 14
 - hardware 225
 - installation 14
 - memory 14
 - software 225
- reset 295
- resolve in-doubt messages 295
- resolve option 295
- resources
 - protected 86
 - why protect 84
- restart queue manager 46
- restart recovery
 - with circular logging 127
- restrictions 83
 - access to MQM objects 83
 - object names 171
- retrieval algorithms for messages 5
- RETRY keyword, rules table 113
- RETRYINT keyword, rules table 109
- return codes 150
 - crtmqcvx command 174
 - crtmqm command 178
 - dltmqm command 180
 - dspmqaout command 184
 - dspmqcsv command 186
 - dspmqfls command 188
 - endmqcsv command 190
 - endmqm command 193
 - endmqtrc command 194
 - interpreting values of 173
 - rcdmqimg command 196
 - rcrmqobj command 198
 - rsvmqtrn command 200
 - runmqchi command 201
 - runmqchl command 202
 - runmqslr command 205
 - runmqsc command 207
 - runmqtrmc command 209
 - runmqtrm command 210
 - setmqaut command 215
 - strmqcsv command 217
 - strmqm command 218
 - strmqtrc command 220
- rights identifier
 - default for authority 85
 - default, nobody 85
 - MQM 83
- rights identifiers, for authority 85
- rsvmqtrn command 199
 - parameters 199
 - related commands 200
 - return codes 200
- rules table, DLQ handler 108
 - See also* DLQ handler
- rules table, DLQ handler (*continued*)
 - control data entry 109
 - INPUTQ keyword 109
 - INPUTQM keyword 109
 - RETRYINT keyword 109
 - WAIT keyword 109
 - example 117
 - patterns and actions (rules) 110
 - ACTION keyword 112
 - APPLIDAT keyword 110
 - APPLNAME keyword 111
 - APPLTYPE keyword 111
 - DESTQ keyword 111
 - DESTQM keyword 111
 - FEEDBACK keyword 111
 - FORMAT keyword 111
 - FWDQ keyword 112
 - FWDQM keyword 112
 - HEADER keyword 113
 - MSGTYPE keyword 111
 - PERSIST keyword 111
 - PUTAUT keyword 113
 - REASON keyword 111
 - REPLYQ keyword 111
 - REPLYQM keyword 111
 - RETRY keyword 113
 - USERID keyword 112
 - processing of 115
 - syntax 113
- run channel 290
- run listener (runmqslr command) 205
- runmqchi command 201
 - parameters 201
 - return codes 201
- runmqchl command 202
 - parameters 202
 - return codes 202
- runmqdlq command 107
- runmqslr command 205
 - parameters 205
 - return codes 205
- runmqsc
 - ending 51
 - feedback 51
 - issuing MQSC commands 50
 - problems 56
 - queued mode 76
 - specifying a queue manager 52
 - using 53
 - using interactively 51
 - verifying 56
- runmqsc command 206
 - examples 207
 - parameters 206
 - redirecting input and output 53
 - return codes 207

- runmqtmc command 209
 - parameters 209
 - return codes 209
- runmqtrm command 210
 - parameters 210
 - return codes 210

S

- sample
 - MQSC files 233
 - programs, using 233
- samples
 - trace data 165
- secondary log files 145
- security 83
 - enabling 86
 - remote 91
 - using the commands 87, 89
- sending on DECnet Phase IV 305
- sending on TCP/IP
 - Digital OpenVMS systems 297
- set/reset authority command
 - See setmqaut command
- setmqaut command 211
 - examples 216
 - installable services 88
 - parameters 213
 - related commands 216
 - return codes 215
 - using 87
- setting up communication
 - Digital OpenVMS systems 297
- shared queues
 - configuration tasks 103
- sharing queues 103
- shutdown
 - queue manager 45
 - controlled 45
 - immediate 45
 - preemptive 45
 - quiesce 45
- SNA configuration
 - Digital OpenVMS systems 300
- SNA Listener process, ending 303
- SO_KEEPALIVE
 - Digital OpenVMS systems 298
 - UNIX systems 298
- Spanish language support 248
- specified operating environment 225
- specifying coded character set 28
- stanzas
 - mqs.ini 139
 - qm.ini 141
- start channel
 - OS/2 290
 - start channel (*continued*)
 - UNIX systems 290
 - Windows NT 290
 - start MQSeries trace command 219
 - start queue manager command 218
 - starting
 - a queue manager 44
 - channels 75
 - command server 47
 - status
 - display channel 290
 - stop force 295
 - stop quiesce 294
 - stopping
 - command server 48
 - queue manager
 - manually 275
 - stopping a channel 294
 - strmqscv command 217
 - examples 217
 - parameters 217
 - related commands 217
 - return codes 217
 - strmqm command 218
 - examples 218
 - parameters 218
 - related commands 218
 - return codes 218
 - strmqtrc command 219
 - examples 220
 - parameters 219
 - related commands 221
 - return codes 220
 - structures
 - initial values for MQSeries for Digital OpenVMS 280
 - supported code sets 28
 - Swedish language support 246
 - syncpoint, performance considerations 157
 - syntax
 - help 172
 - syntax diagrams, how to read 171
 - syntax error, in MQSC commands 51
 - SYSD\$INPUT, on runmqsc 53
 - SYSD\$OUTPUT, on runmqsc 53
 - system
 - objects
 - defining 29
 - system configuration 21
 - system default objects 10
 - system defaults 227
 - system limitations 21
 - system objects
 - creating 44

T

- TCP/IP connection
 - MQSeries for Digital OpenVMS systems 297
- TCP/IP KEEPALIVE
 - Digital OpenVMS systems 298
 - UNIX systems 298
- temporary queues 5
- terminology used in this book 349
- time-independent applications 3
- timed out responses from MQSC commands 76
- trace
 - data sample 165
 - performance considerations 164
- transactions
 - resolve MQSeries command 199
- translated messages 22
- transmission queue 80
 - creating 80
 - default 41, 80
 - defining 74
 - description 8
 - remote administration 73
- trigger
 - event queues 122
 - events
 - compared with instrumentation events 122
 - messages on initiation queue 8
 - monitor
 - description 8
 - start command 210
- triggering
 - application queue
 - defining 68
 - managing objects for 68
- trusted applications 237
 - channel programs 238
 - restrictions 239
- typedef
 - for MQSeries for Digital OpenVMS 280
- types of event 121
- types of objects 5

U

- UK English language support 249
- unauthorized access, protecting from 84
- undelivered message queue
 - See dead-letter queue
- updating coded character sets 28
- US English language support 243
- user ID
 - authority 83
 - authorization 89
 - belonging to group nobody 85
 - for authorization 89

- user ID (*continued*)

- OpenVMS logged-in user 89
- USERID keyword, rules table 112
- users
 - identifiers
 - principals 85
- using copy files
 - on MQSeries for Digital OpenVMS 280

V

- verifying MQSC commands 56

W

- WAIT keyword, rules table 109
- Windows clients error messages 167

Sending your comments to IBM

MQSeries for Digital OpenVMS

System Management Guide

GC33-1791-00

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: WINVMD(IDRCF)
 - Internet: idrcf@winvmd.vnet.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 for Digital OpenVMS

System Management Guide

GC33-1791-00

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	

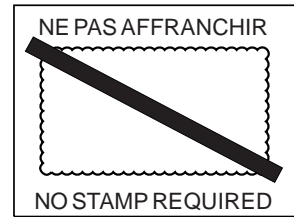
If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCR NUMBER: PHQ - D/1348/SO



**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 _____

1 Cut along this line

4 Fasten here with adhesive tape



Printed in U.S.A.

GC33-1791-00

